Angular Demos
Core Dynamic Features

{{ expr }}, *ngFor, *ngIf
@Component … @Input @Output
(upgraded to Angular 17)

arnaud.nauwynck@gmail.com

Course Esilv 2024

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

Outline

templateHtml {{ expr }}
*ngFor, @for() { }
*ngIf, @if() { }
@Component
[] .. @Input field, input()
() .. @Output changed, output()
[()] .. Bidirectional Binding, model()
router

Outline

templateHtml {{ expr }}

*ngFor, @for() { }

*ngIf, @if() { }

@Component

[] .. @Input field, input()

() .. @Output changed, output()

[()] .. Bidirectional Binding, model()

router

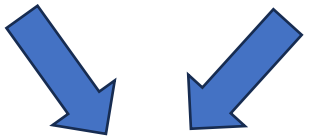
Interpolating {{ expr }} in template Html

In component**.ts**

```ts
@Component({
  selector: 'app-demo2-interpolation',
  templateUrl: './demo2-interpolation.component.html',
})
export class Demo2InterpolationComponent {

  value1Text = "text1";

}
```

In (template) component**.html**

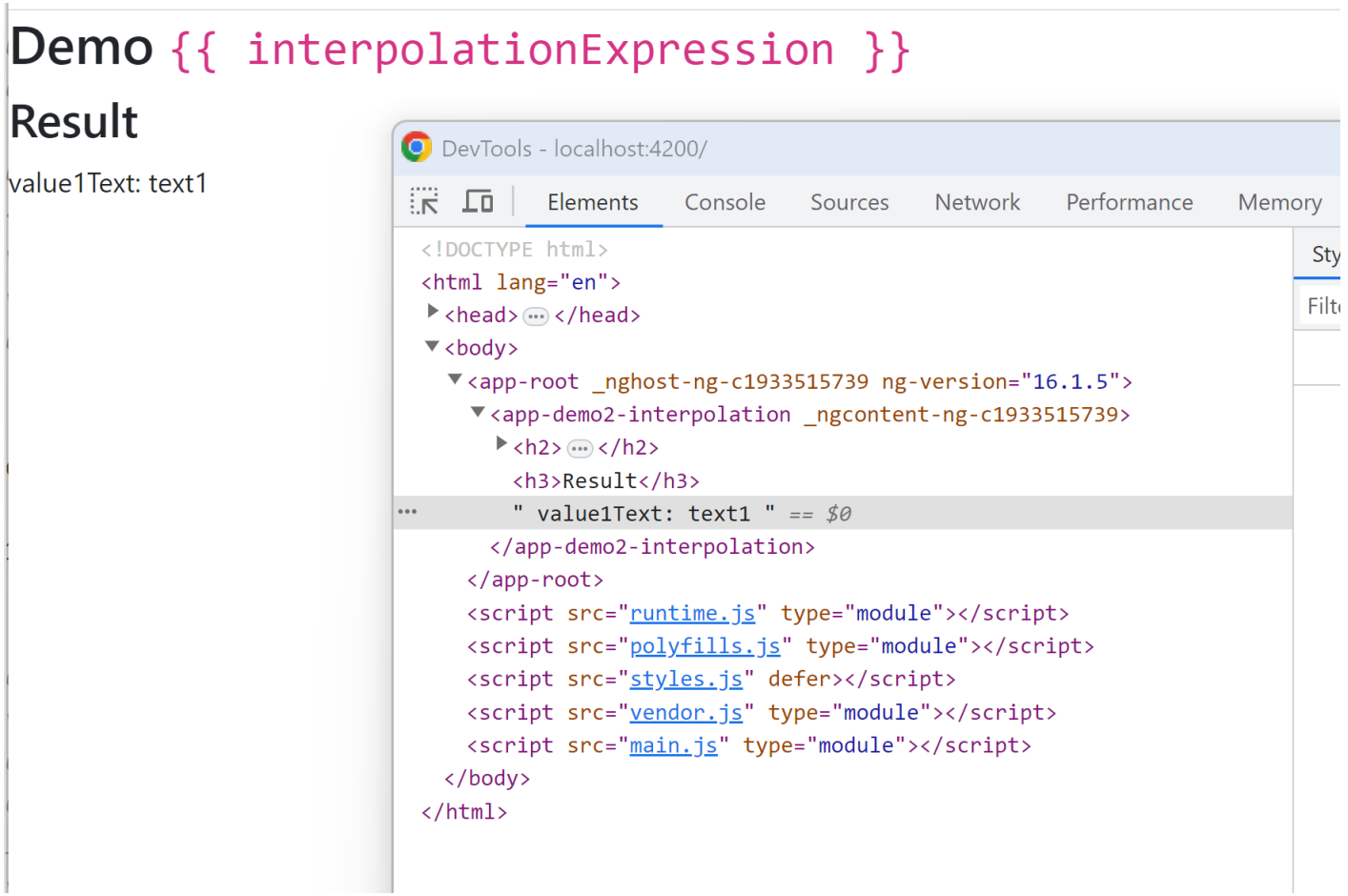
```
value1Text: {{ value1Text }}
```



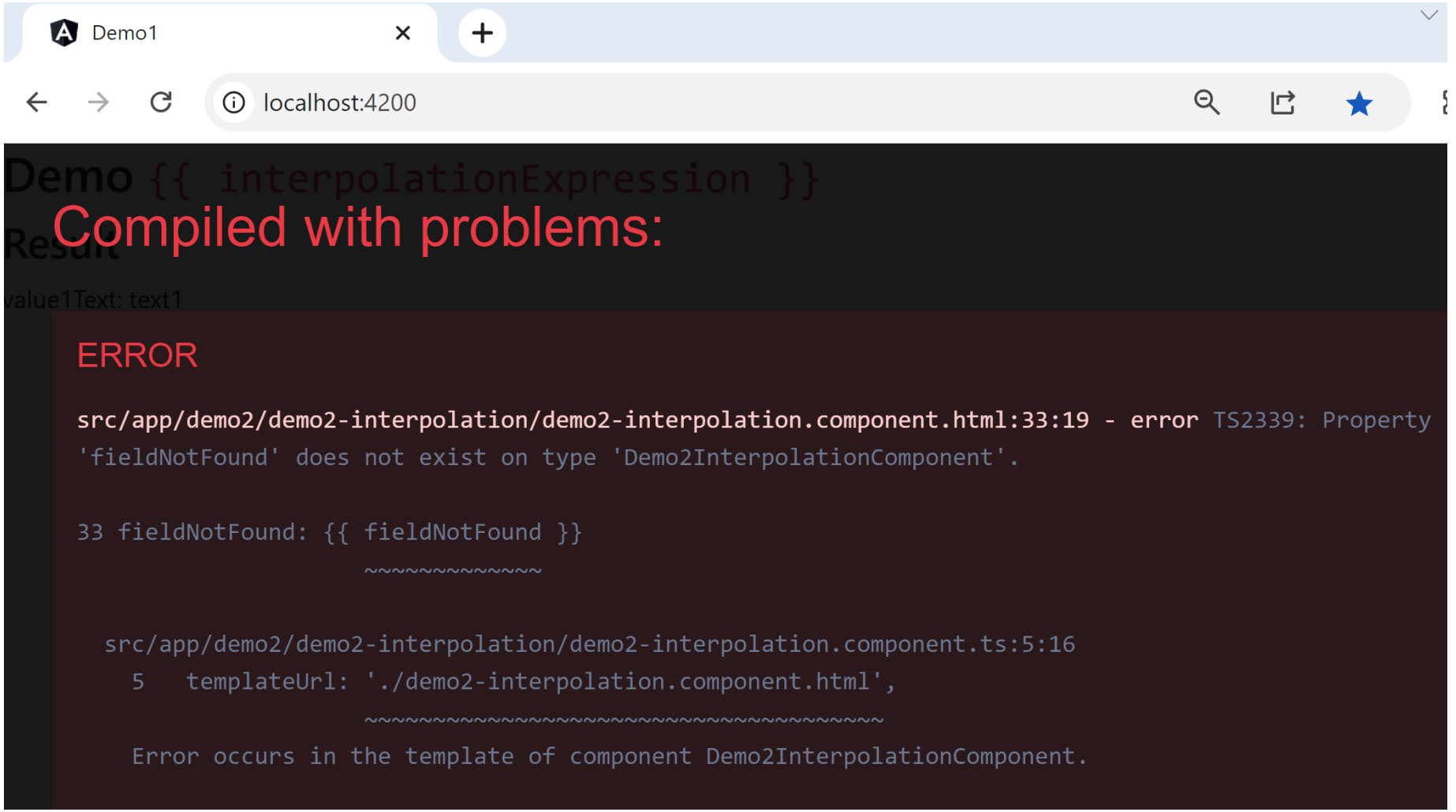
Result DOM

value1Text: text1

Checking with Chrome debugger (F12)



What if {{ fieldNotFound }} ?



Demo {{ complexExpr }}

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

value1Number = 12;
value2Number = 3.14159;

value1Bool = true;
value2Bool = false;

value1Obj = { text: "objectText1", fieldInt: 1 };

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

function1(): string { return this.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, @for() { }

*ngIf, @if() { }

@Component

[] .. @Input field, input()

() .. @Output changed, output()

[()] .. Bidirectional Binding, model()

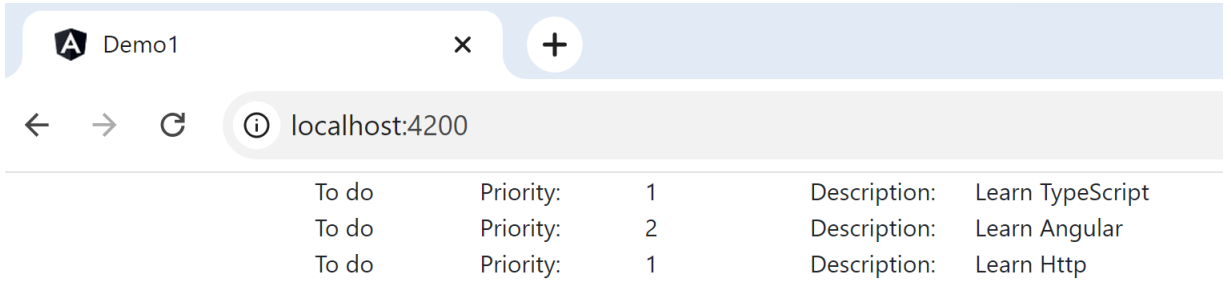
router

Demo *ngFor

Expected html …
but without copy&paste

app.component.html

```
<div class="container">
  <div class="row">
    <label class="col-md-1">To do</label>
    <label class="col-md-1">Priority: </label>
    <span class="col-md-1">1</span>
    <label class="col-md-1">Description: </label>
    <span class="col-md-8">Learn TypeScript</span>
  </div>
  <div class="row">
    <label class="col-md-1">To do</label>
    <label class="col-md-1">Priority: </label>
    <span class="col-md-1">2</span>
    <label class="col-md-1">Description: </label>
    <span class="col-md-8">Learn Angular</span>
  </div>
  <div class="row">
    <label class="col-md-1">To do</label>
    <label class="col-md-1">Priority: </label>
    <span class="col-md-1">1</span>
    <label class="col-md-1">Description: </label>
    <span class="col-md-8">Learn Http</span>
  </div>
</div>
```



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

app.component.ts

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

interface TodoItem {
  priority: number;
  description: string;
}

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'demo1';

  items: TodoItem[] = [
    { priority: 1, description: 'learn TypeScript' },
    { priority: 2, description: 'learn Angular' },
    { priority: 1, description: 'learn Http' },
    ];
```

app.component.html

```html
<div class="container">
  <div class="row" *ngFor="let item of items">
    <label class="col-md-1">To do</label>
    <label class="col-md-1">Priority: </label>
    <span class="col-md-1">{{item.priority}}</span>
    <label class="col-md-1">Description: </label>
    <span class="col-md-8">{{item.description}}</span>
  </div>
</div>
```

*ngFor

Local variable aliases:
Index, count, first / last, even / odd

trackBy:
For incremental optims and transitions

Angular ≥ 17

@for(item of list; track item.id) { ... }

todo-list.component.html

```
<H1>List of TODOs</H1>

<div class="container">
  @for(item of todoService.items; track item.id) {
    <div class='row'>
      <label class='col-md-1'>To do</label>
      <label class='col-md-1'>Priority: </label>
      <span class='col-md-1'>{{item.priority}}</span>
      <label class='col-md-1'>Description: </label>
      <span class='col-md-5'>{{item.description}}</span>
    </div>
  }
</div>
```

Angular ≥ 17

imports: [NgForOf]

todo-list.component.ts

```ts
import { Component } from '@angular/core';
import { NgForOf } from "@angular/common";
import { TodoService } from "../todo.service";

2 usages
@Component({
  selector: 'app-todo-list',
  standalone: true,
  imports: [ NgForOf ],
  templateUrl: './todo-list.component.html',
})
export class TodoListComponent {
```

Outline

templateHtml {{ expr }}

*ngFor, @for() { }

*ngIf, @if() { }

Chrome Debugger

@Component

[] .. @Input field, input()

() .. @Output changed, output()

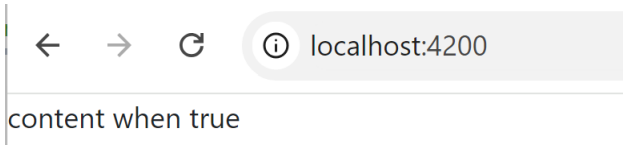
[()] .. Bidirectional Binding, model()

router

*ngIf

```
condition: boolean = true;
```

```
<H2>demo showing *ngIf</H2>

<div *ngIf="condition">
  <div>content when true</div>
</div>
```



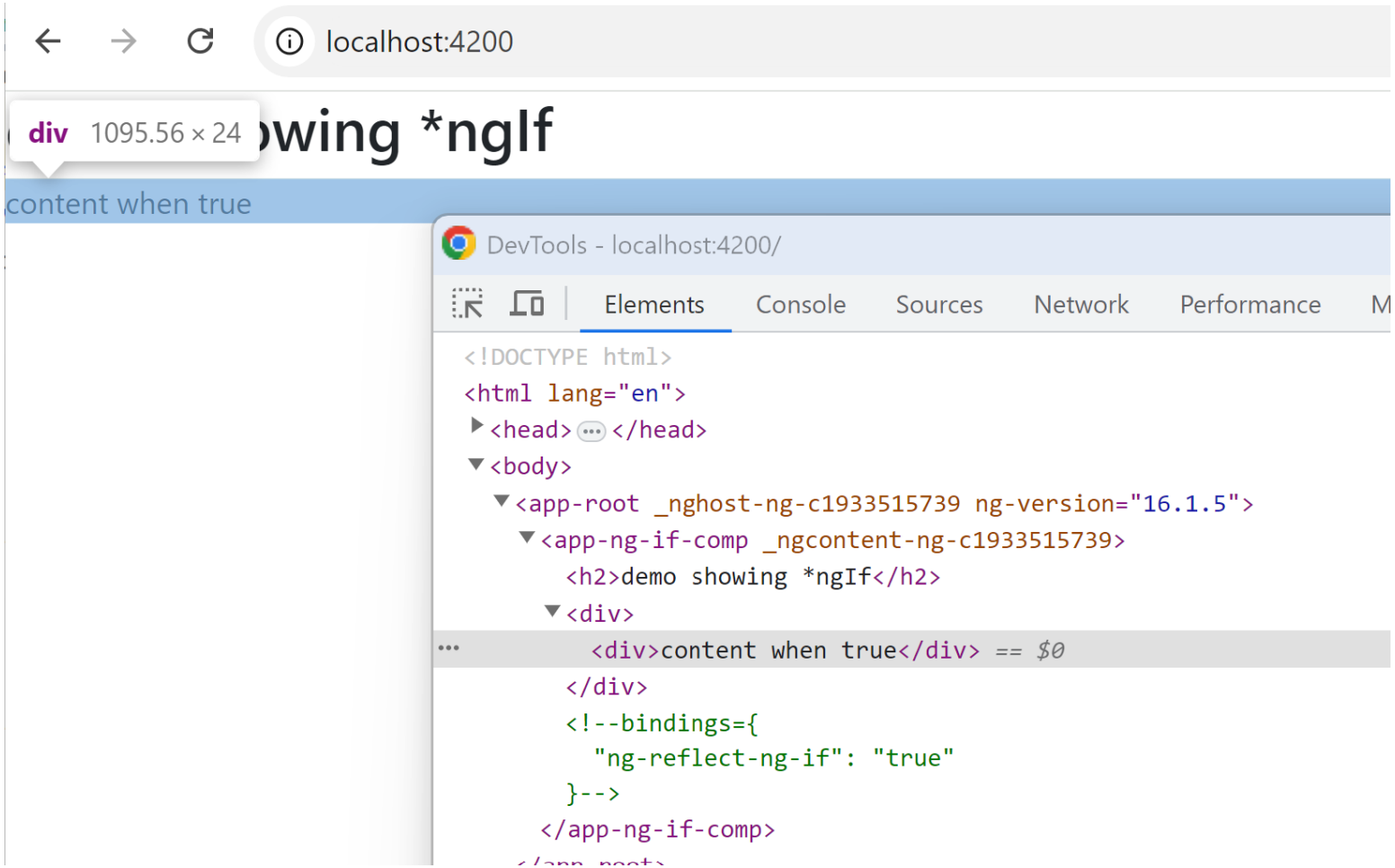
Angular >= 17

@if( ) { } @else { }

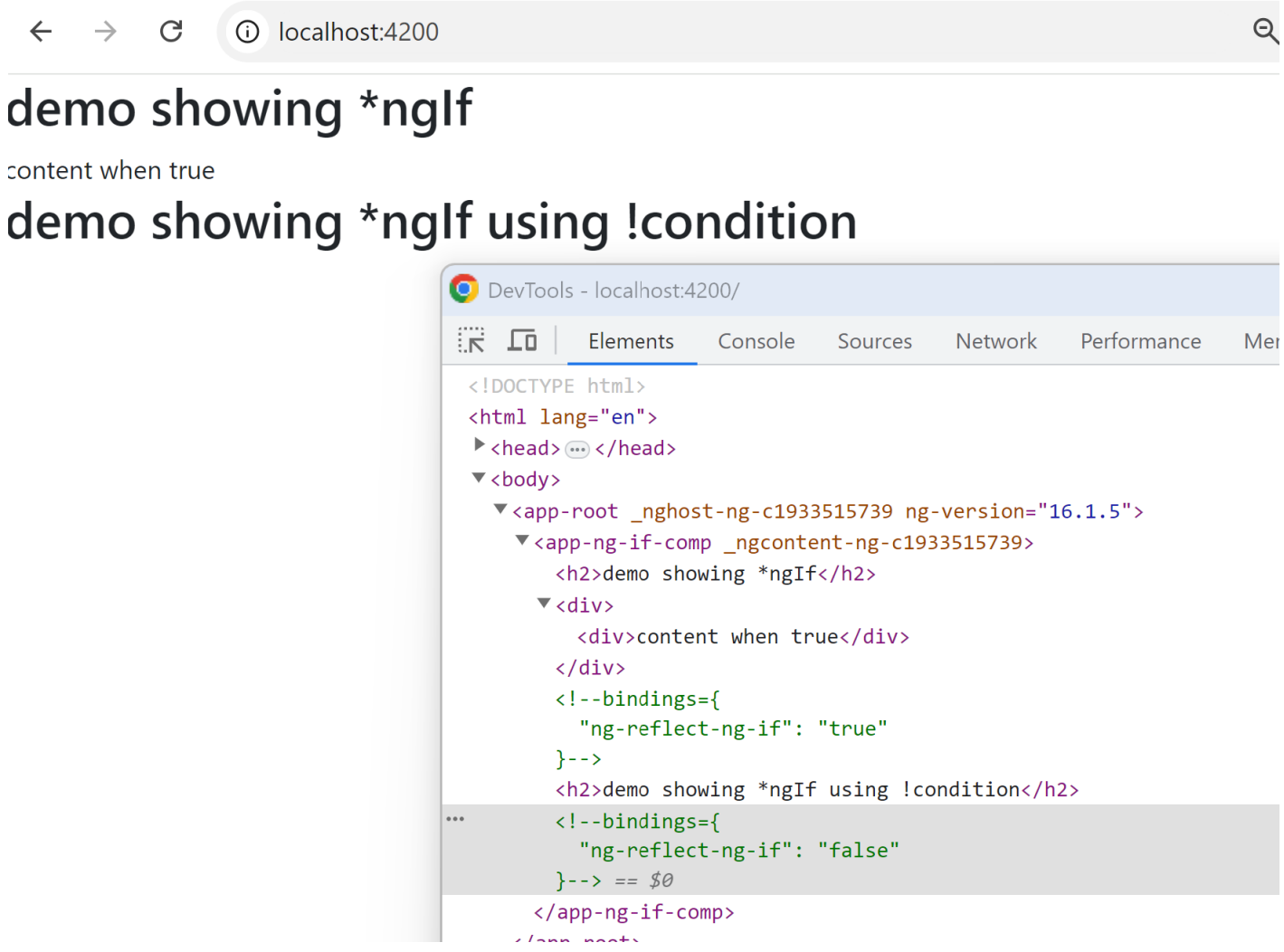
test-if.component.html

```
1  <H2>demo *ngIf</H2>
2
3  @if(condition) {
4    <div>content when true</div>
5  } @else {
6    <div>content when false</div>
7  }
8
9  <button (click)="condition =!condition">Toggle</button>
```

Result *ngIf
=> attribute removed, comment added



*ngIf when false => NO content



*ngIf … ; else contentTemplate => ng-template

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

Solution 2 : use ng-template:

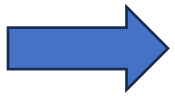
```
<div *ngIf="condition; else templateWhenFalse">
  <div>content when true</div>
</div>
<ng-template #templateWhenFalse>
  <div>content when false</div>
</ng-template>
```

Solution 3 : use *ngSwitch

How to avoid extra on *ngIf with multiple elements content ?

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

Actual content:

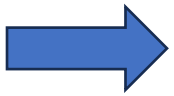


```
<span>
  <div>
    <span>element 1</span>
    <span>element 2</span>
  </div>
</span>
```

Possible problems:
CSS, divs overhead

How to ???

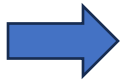
INSTEAD… Expected content:



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

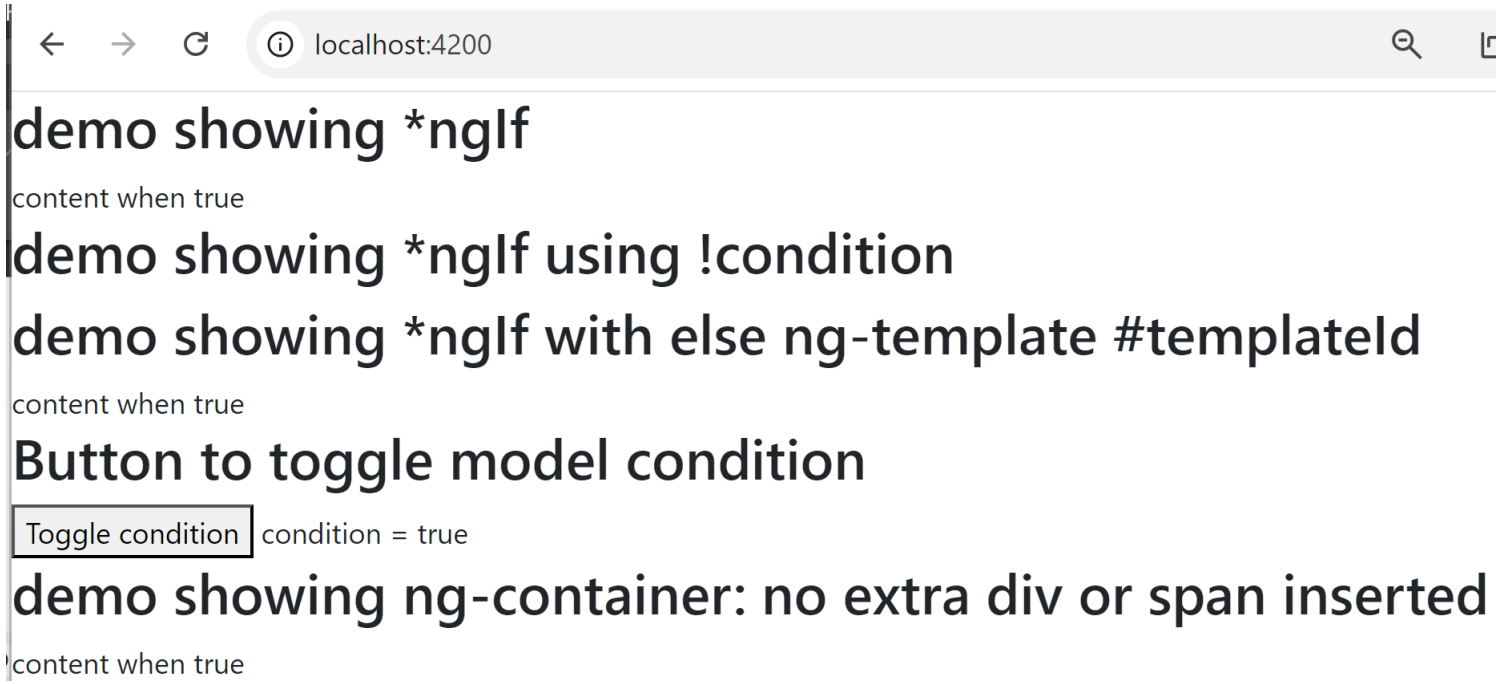
Using <ng-container *ngIf=..>

```
<span>
  <ng-container *ngIf="condition">
    content when true
  </ng-container>
</span>
```



```
<span>
  " content when true " == $0
  <!---->
  <!---->
</span>
```

Demo *ngIf

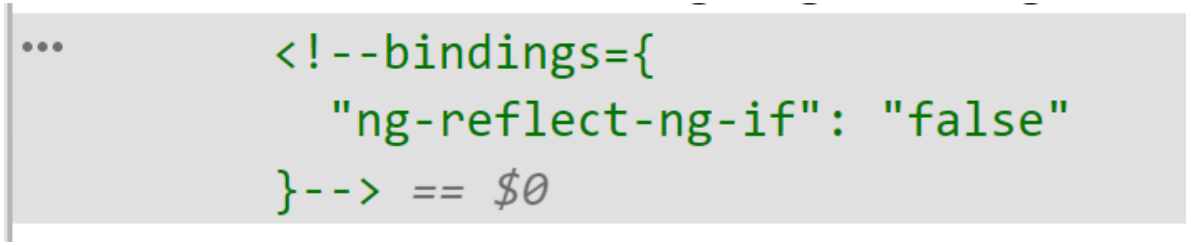

localhost:4200
demo showing *ngIf
content when true
demo showing *ngIf using !condition
demo showing *ngIf with else ng-template #templateId
content when true
Button to toggle model condition
Toggle condition
condition = true
demo showing ng-container: no extra div or span inserted
content when true

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

Default conf « development » :

$ ng serve
(implicit: ng serve –c development)

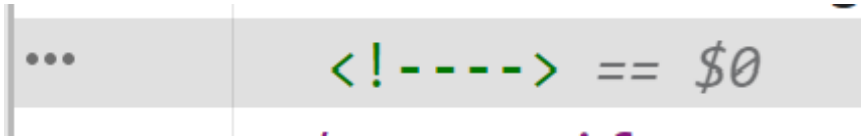
=> Has Chrome debugging comment



conf « production » :

$ ng serve -**c production**

=> Still empty placeholder comment
but NO debugging info in it



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:
      --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 https://angular.io/guide/workspace-config#alternate-build-configurations.
                                                                                [string] [choices: "development", "production"]
      --allowed-hosts         List of hosts that are allowed to access the dev server.                                [array]
```

angular.json « development» vs « production » conf

Outline

templateHtml {{ expr }}
*ngFor, @for() { }
*ngIf, @if() { }

@Component
[] .. @Input field, input()
() .. @Output changed, output()
[()] .. Bidirectional Binding, model()
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)

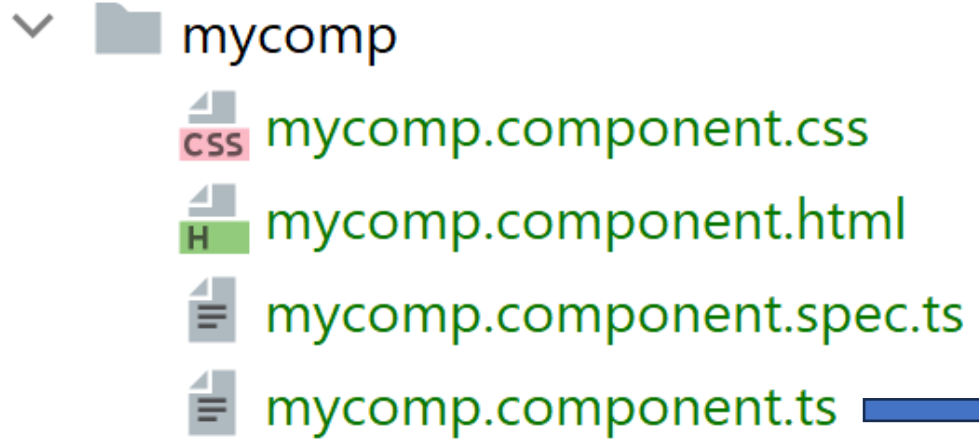
$ ng generate component

(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



+ **2 lines** inserted in module
(cf next !)

app.module.ts

mycomp.component.html

```
1  <p>mycomp works!</p>
```

mycomp.component.ts

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-mycomp',
5    templateUrl: './mycomp.component.html',
6    styleUrls: ['./mycomp.component.css']
7  })
8  export class MycompComponent {
9
10 }
```

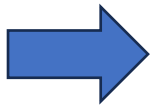
Using @Component
… similar to html « Custom Element »

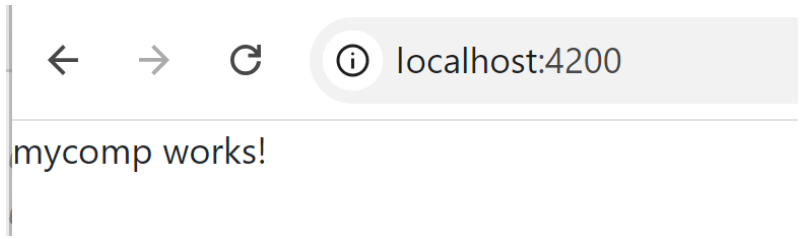
Declare component =
assign selector (element name) **to class**

```
@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!)

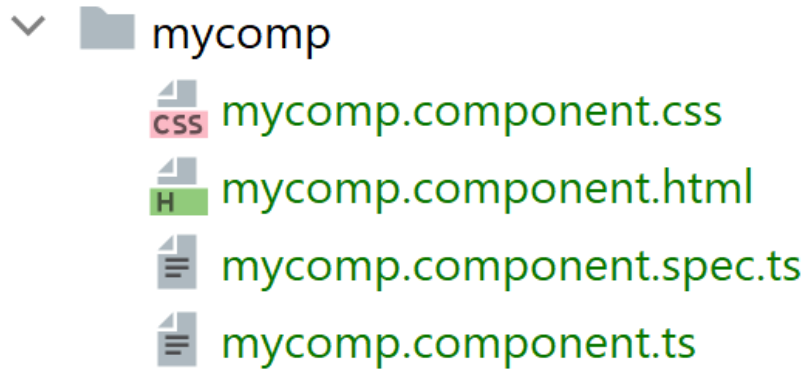
Short @Component declaration

mycomp.component.ts

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

@Component({
  selector: 'app-mycomp',
  template: `<p>mycomp works!</p>`
})
export class MycompComponent {

}
```

@Component…
ADD in @NgModule declarations: […] !!!

1 dir + 4 new files added



+ 2 lines inserted
in existing module !

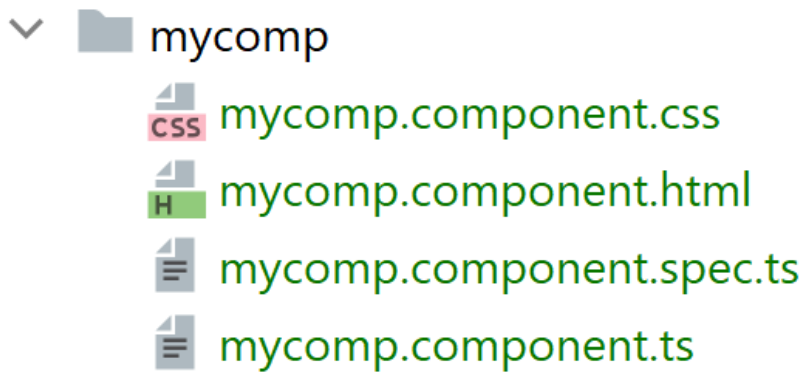
app.module.ts

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

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

Angular >= 17

@Component standalone=true

1 dir + 4 new files added



my-comp.component.ts

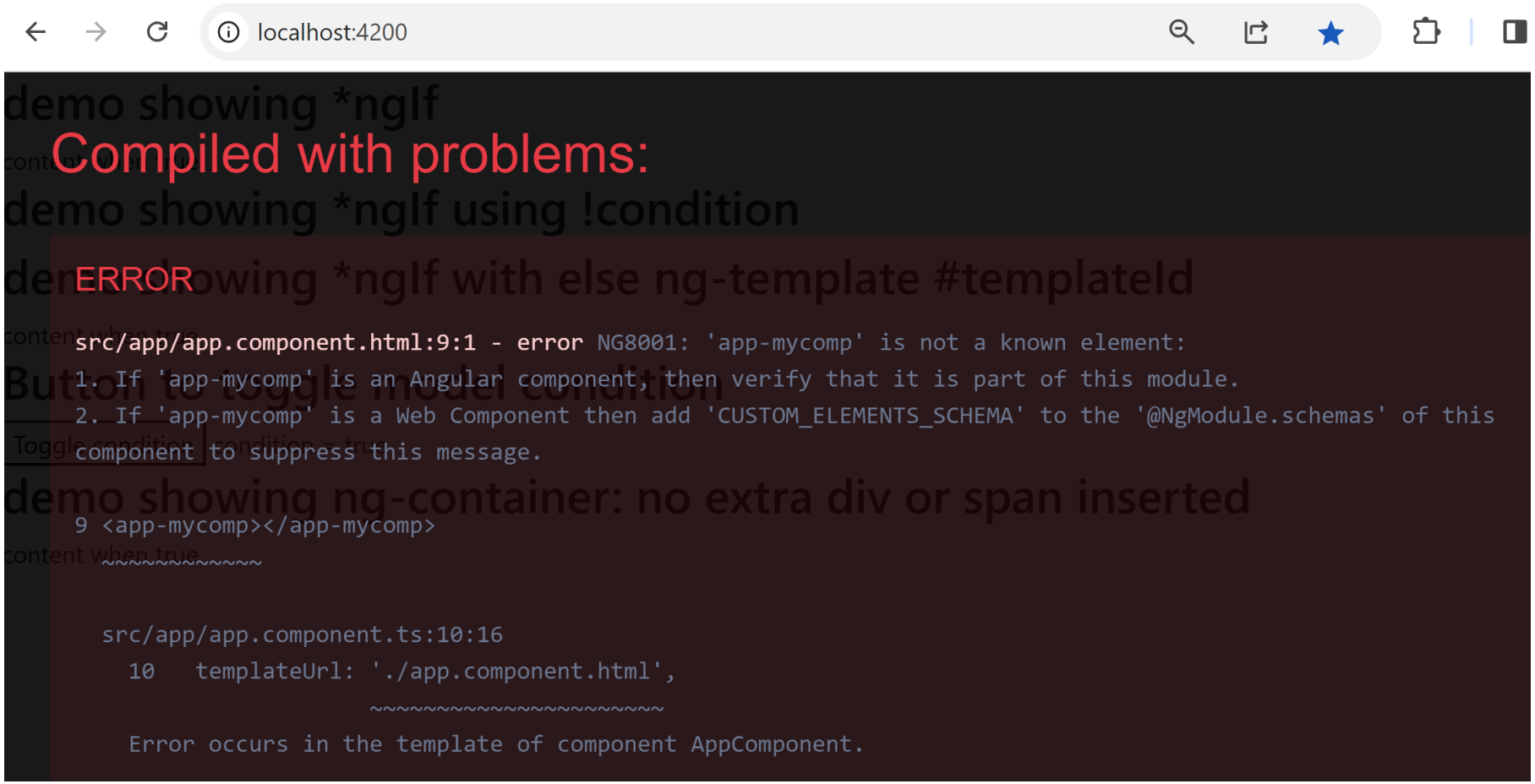
```ts
import { Component } from '@angular/core';

5+ usages
@Component({
  selector: 'app-my-comp',
  standalone: true,
  imports: [],
  templateUrl: './my-comp.component.html',
  styleUrl: './my-comp.component.css'
})
export class MyCompComponent {

}
```

no module !

no need to add in config

app.config.ts

Using component, but forgetting to add in module



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

$\Rightarrow$ Avoid modifying too many git commits / conflicts in same file

Common practices :

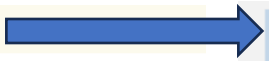
Solution 1/ split module in sub-modules

Solution 2/ split array declarations into constants sub-arrays

Split declarations with array (...spreading) const

demo2-features.ts

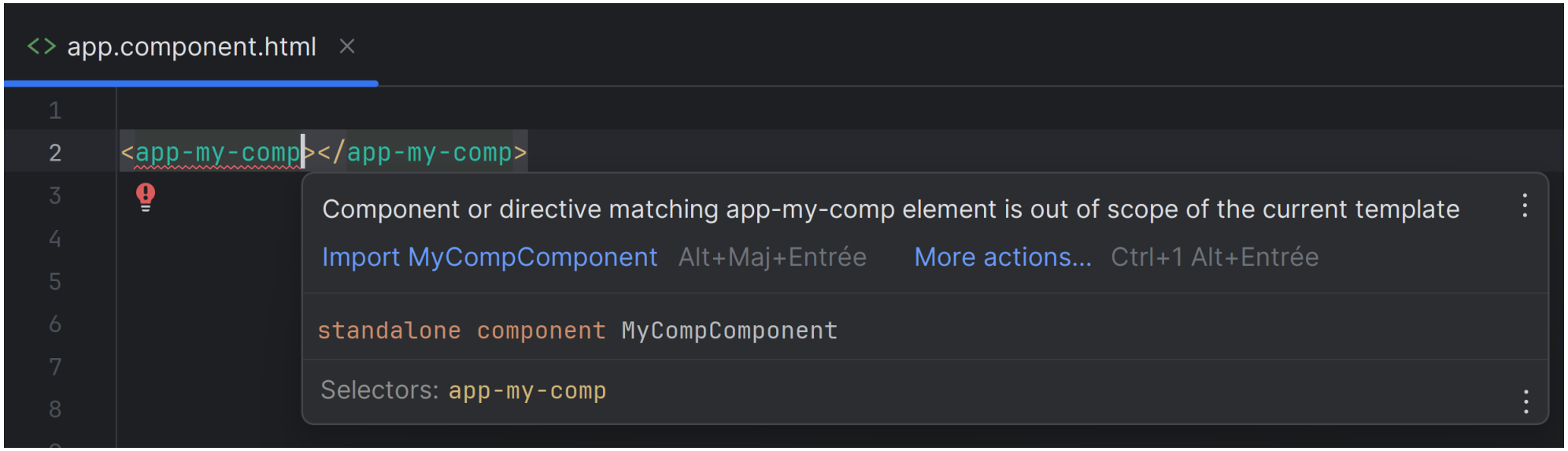
```ts
import { Comp1Component } from './comp1/comp1.component';
import { Demo2InterpolationComponent } from './demo2-interpolat
import { NgIfCompComponent } from './ng-if-comp/ng-if-comp.comp
import { DemoCompInputComponent } from './demo-comp-input/demo-
import { DemoCompInputChildComponent } from './demo-comp-input/

export const DEMO2_COMPONENTS = [
  Demo2InterpolationComponent,
  NgIfCompComponent,
  MycompComponent,
  DemoCompInputComponent,
  Comp1Component,
  DemoCompInputComponent, DemoCompInputChildComponent,
];
```



```ts
import { DEMO2_COMPONENTS } from './demo2/demo2-features';

@NgModule({
  declarations: [
    AppComponent,
    ...DEMO2_COMPONENTS,
  ],
  imports: [
```

Angular ≥ 17

adding <app-my-comp />



Angular >= 17

import { MyComp } from '..';
@Component({ imports=[MyComp], ..

my-comp-parent.component.ts

```ts
import { Component } from '@angular/core';
import {MyCompComponent} from "../my-comp/my-comp.component";

5+ usages
@Component({
  selector: 'app-my-comp-parent',
  standalone: true,
  imports: [
    MyCompComponent
  ],
  templateUrl: './my-comp-parent.component.html',
  styleUrl: './my-comp-parent.component.css'
})
export class MyCompParentComponent {
```

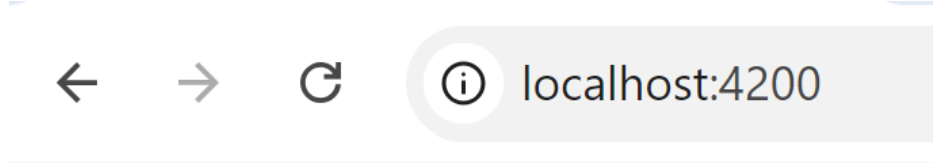
Outline

templateHtml {{ expr }}
*ngFor, @for() { }
*ngIf, @if() { }
@Component

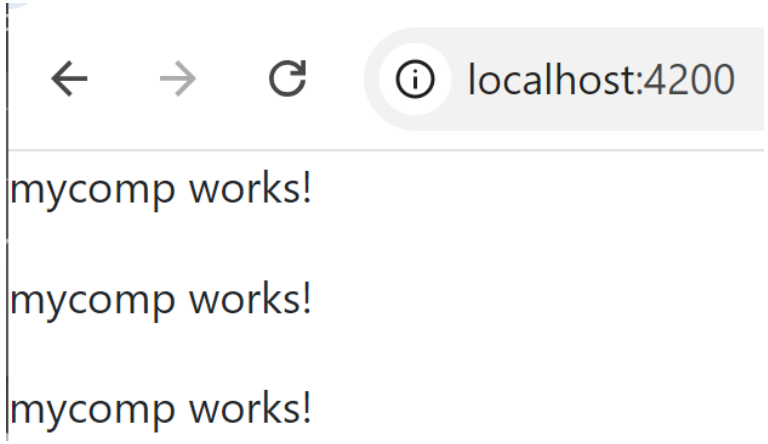
[] .. @Input field, input()
() .. @Output changed, output()
[()] .. Bidirectional Binding, model()
router

Using repeated (static) components … need dynamic binding

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




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



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

```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

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

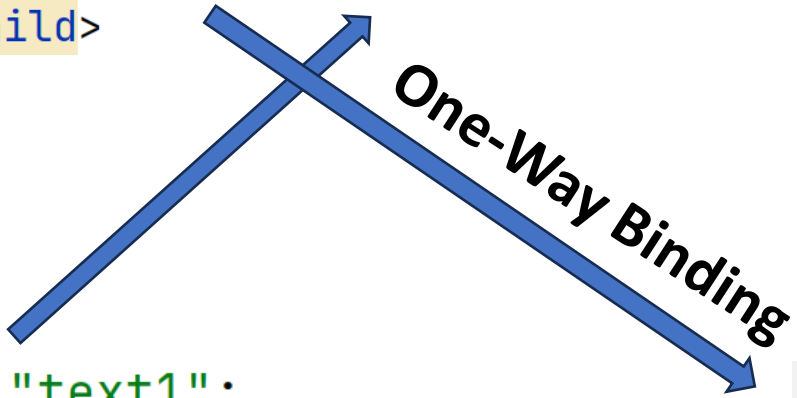
passing parent value to child component: <child [childField]=« parentValue »></child>

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 = '';
```

Angular ≥ 17

instead of "@Input() field: string"
field = input()

test-input.component.ts

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

5+ usages
@Component({
  selector: 'app-test-input',
  standalone: true,
  imports: [],
  templateUrl: './test-input.component.html',
})
export class TestInputComponent {

  field : InputSignal<string>  = input( initialValue: '');

}
```

Angular >= 17

:InputSignal<Type> instead of :Type

field is NOT modifiable from component
=> it is a "read-only getter" method
to use it, call "field()"

test-input.component.html

```
1
2   field:  {{field()}}
```

when forgetting parenthesis "()"

```
5   error: {{field}}
```

=> get

field: hello

error: [Input Signal: hello]

```
▲ [WARNING] NG8109: field is a function and should be invoked: field() [plugin angular-compiler]

    src/app/test-input/test-input.component.html:11:9:
      11 │ error: {{field}}
         ╵          ~~~~~
```

Angular >= 17

field = input(initialValue)

field can have initial value
=> type is implicitly infered

```
field2 : InputSignal<string>  = input( initialValue: '');
```

text in grey is additional display by IntelliJ, but not in file ! real file content :

field2 = input('');

Angular ≥ 17

field = input.required<Type>()

field can be "required"

```
field3 : InputSignal<string>   = input.required<string>();
```

when forgetting to set [field3]='...'

```
<app-test-input></app-test-input>
```

```
X [ERROR] NG8008: Required input 'field3' from component TestInputComponent must be specified. [plu
gin angular-compiler]

    src/app/test-input-parent/test-input-parent.component.html:2:0:
      2 | <app-test-input></app-test-input>
        | ~~~~~~~~~~~~~~~~
```

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

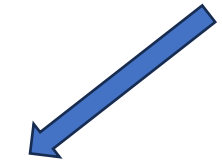
Outline

templateHtml {{ expr }}
*ngFor, @for() { }
*ngIf, @if() { }
@Component
[] .. @Input field, input()

() .. @Output changed, output()
[()] .. Bidirectional Binding, model()
router

Using button callback: (click)=« code…»

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



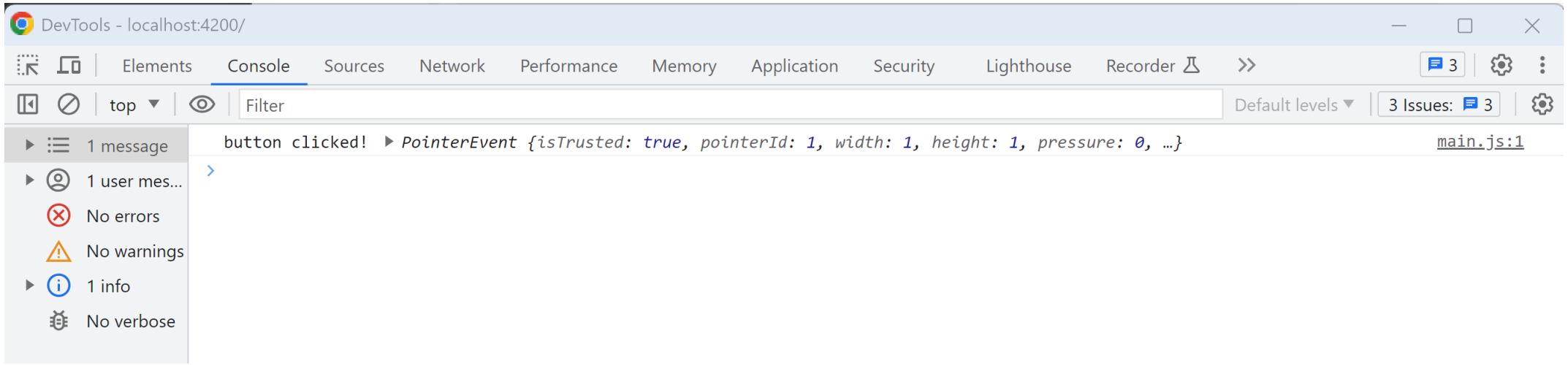
```
onClick() {
  console.log("button clicked!");
}
```

(click) using implicit parameter « $event »

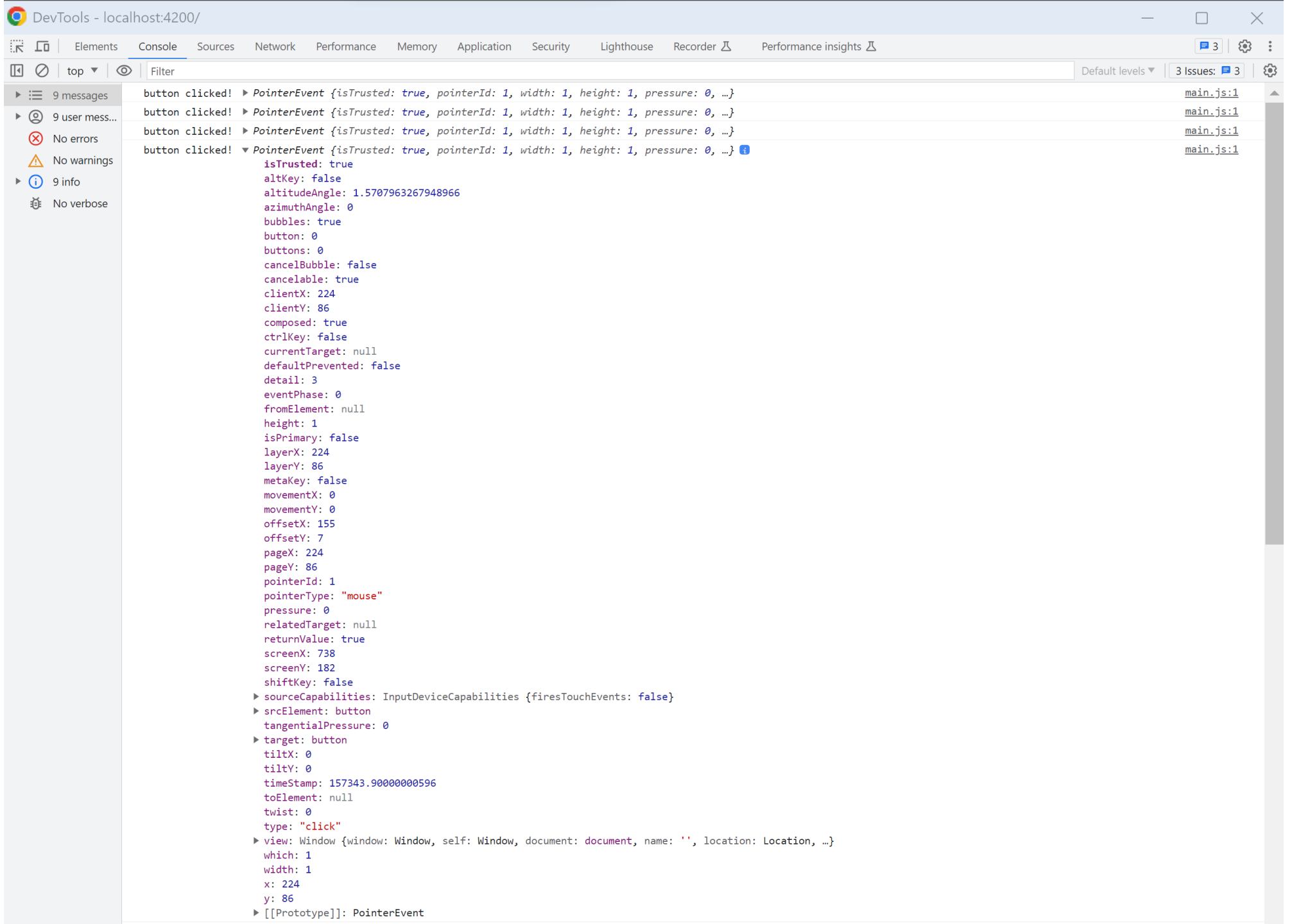
```html
<button (click)="onClick($event)">Click</button>
```

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

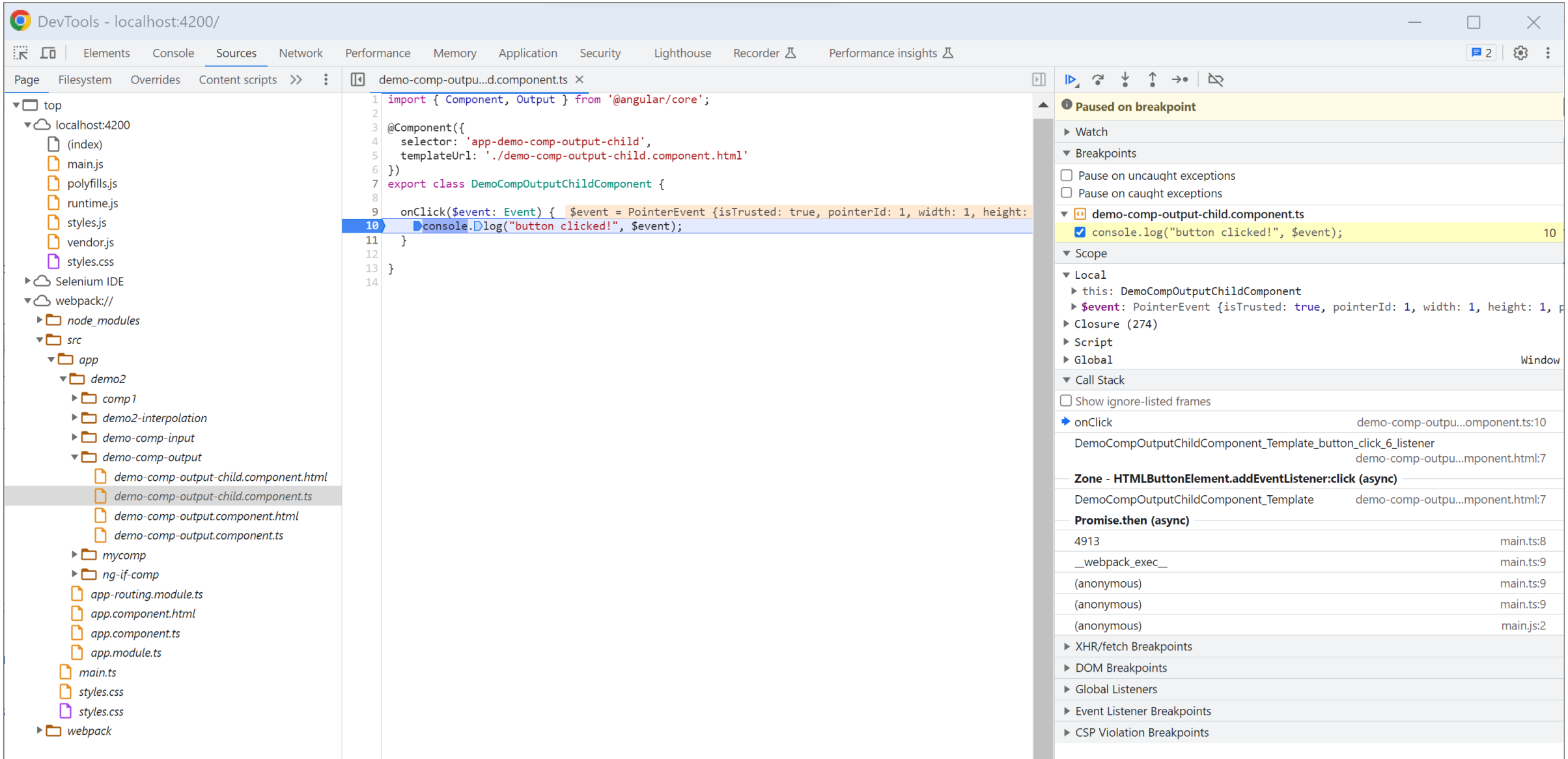
Click Demo … log in Chrome F12 > Console



$event detail

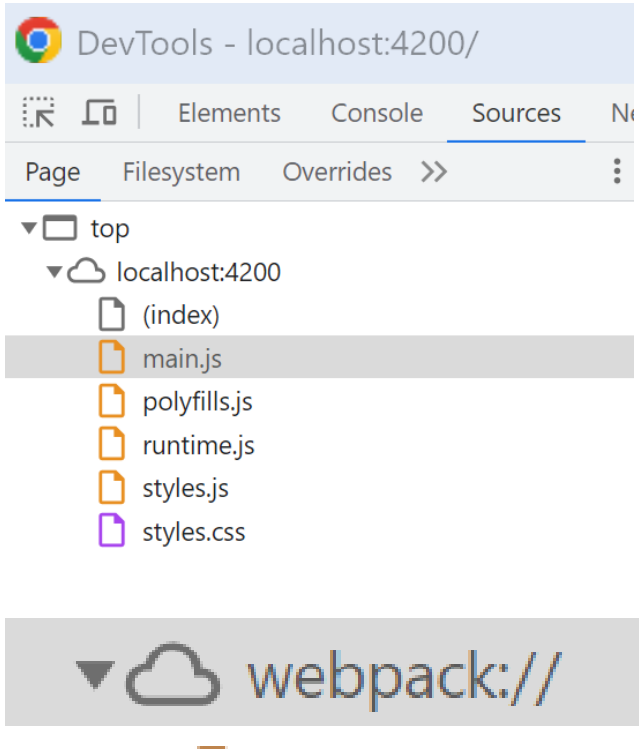


Chrome Debug breakpoint



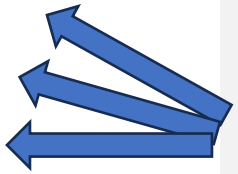
Notice .. « ng serve –c production »
=> NO source « .map » code !
Can not browse to put breakpoint (obfuscated code)

Remark
Source .map



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

@Output() = Observable subscribed from parent components



```typescript
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  }
```

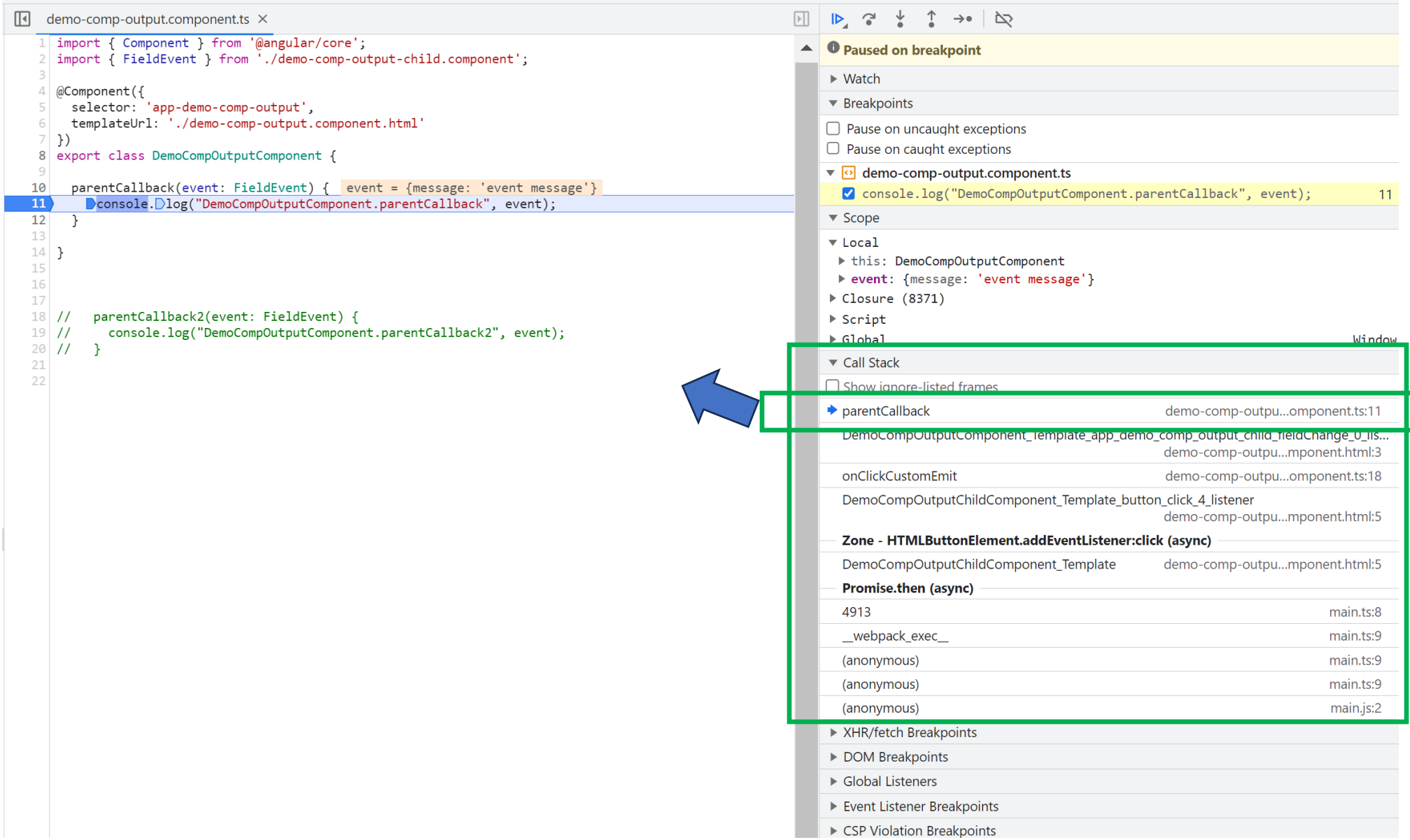
Angular ≥ 17

field = output<Type>()

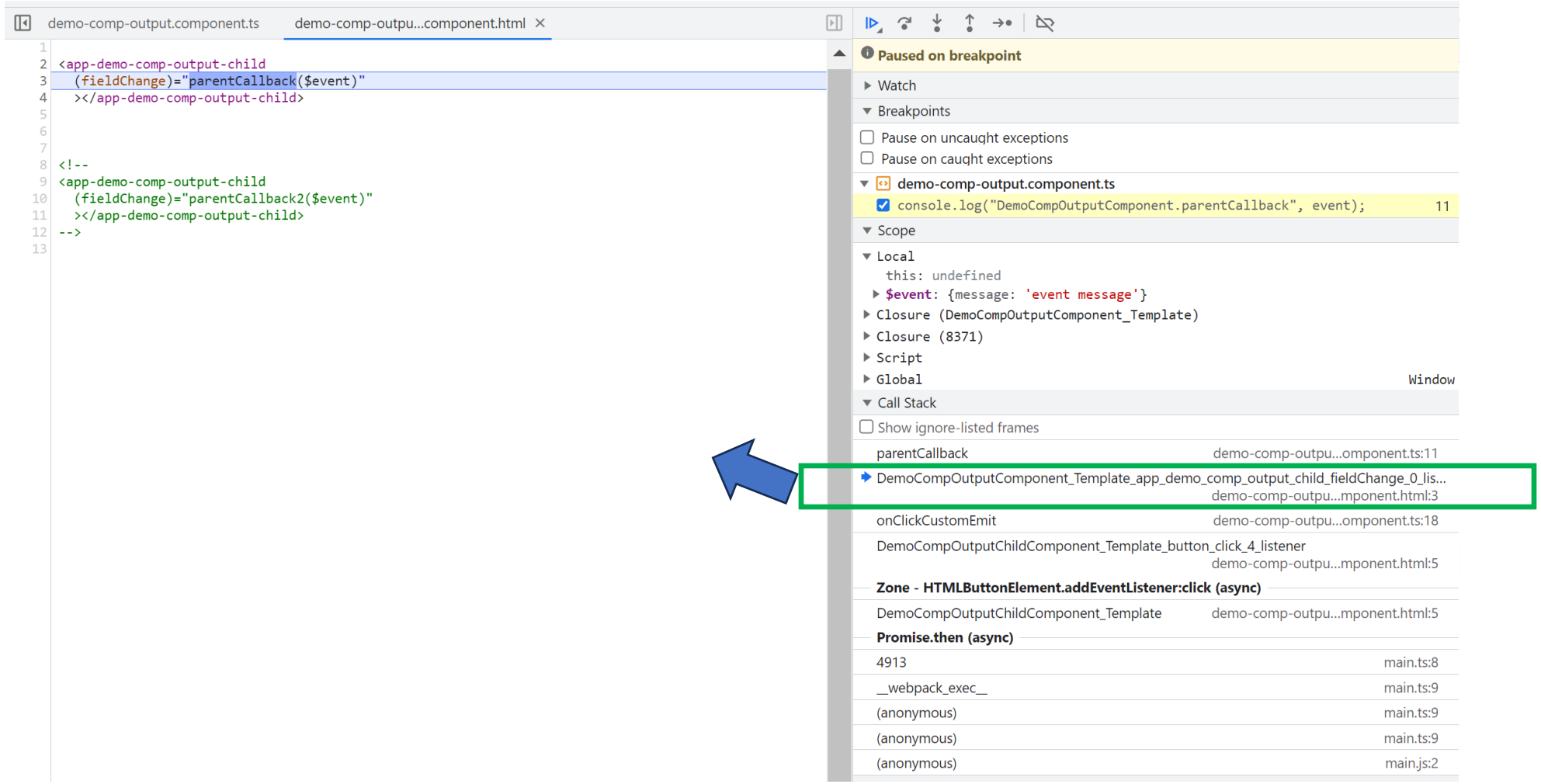
test-output.component.ts

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

5+ usages
@Component({
  selector: 'app-test-output',
  standalone: true,
  imports: [],
  templateUrl: './test-output.component.html',
})
export class TestOutputComponent {

  field : OutputEmitterRef<string>  = output<string>();

  1 usage
  onClick() : void  {
    console.log('click child => emit field');
    this.field.emit( value: 'emit event');
  }
```

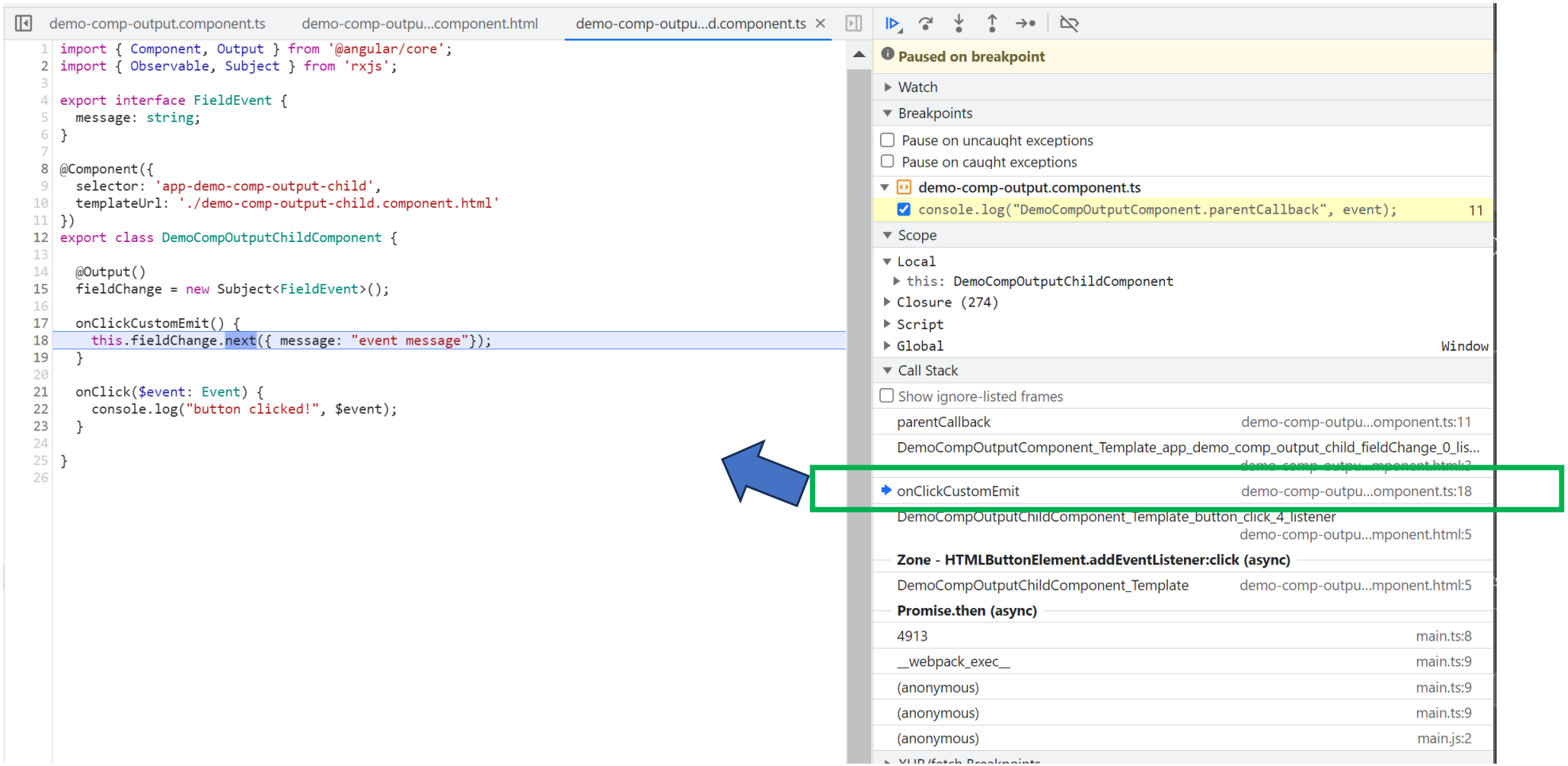
Chrome Debugger > « Call Stack » (parent ts)



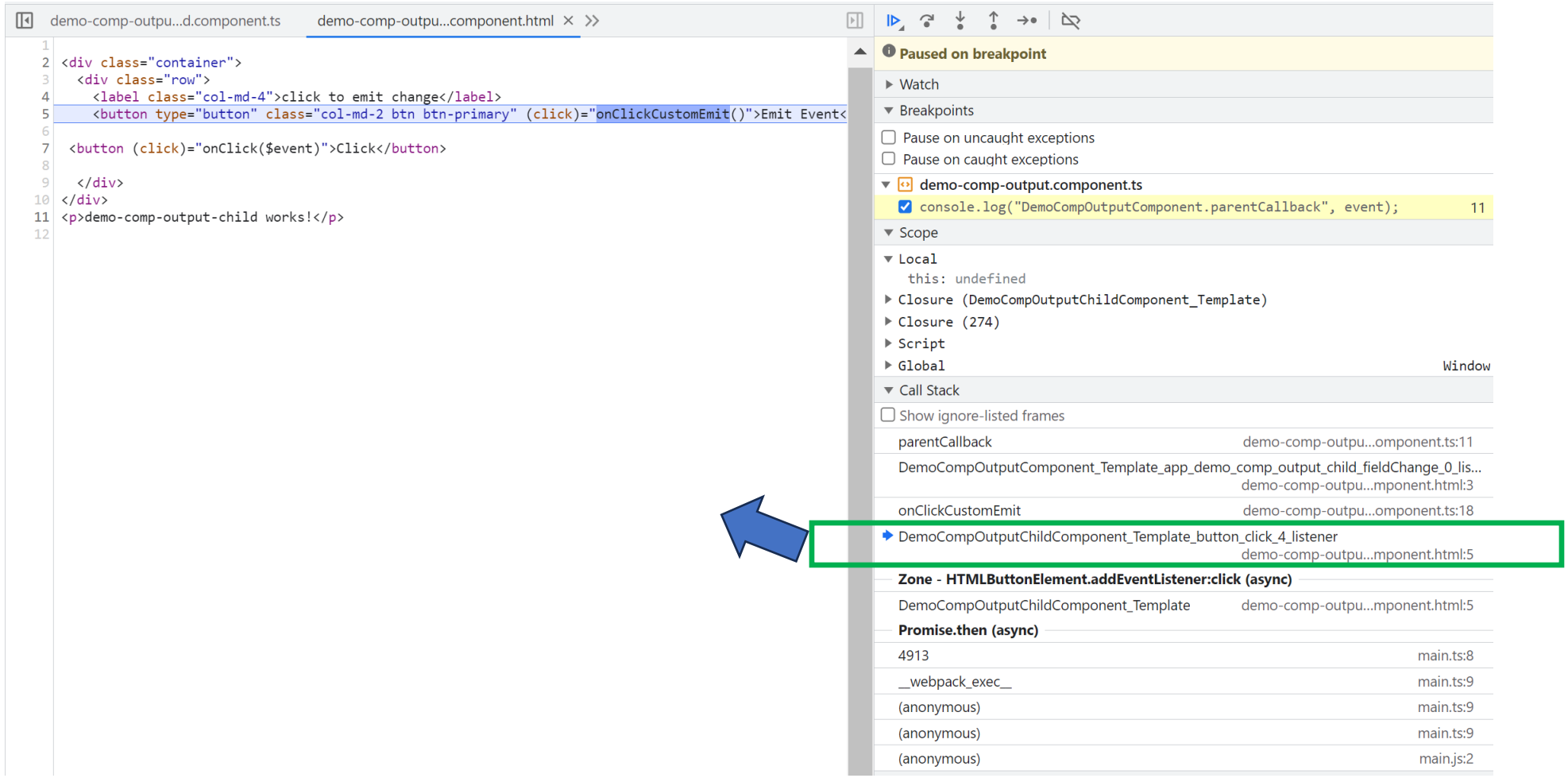
Debugger Call Stack .. Parent caller[1] (parent html)



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



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



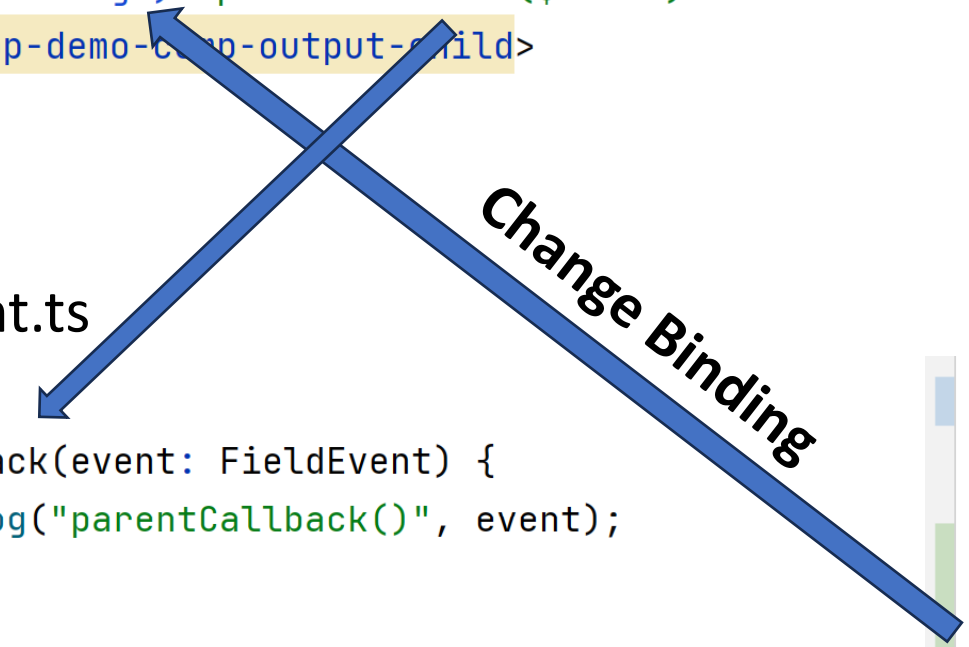
() = Change binding
reverse way: child -> parent

Parent component.html

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

Child component.html

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



Parent component.ts

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

Child component.ts

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

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

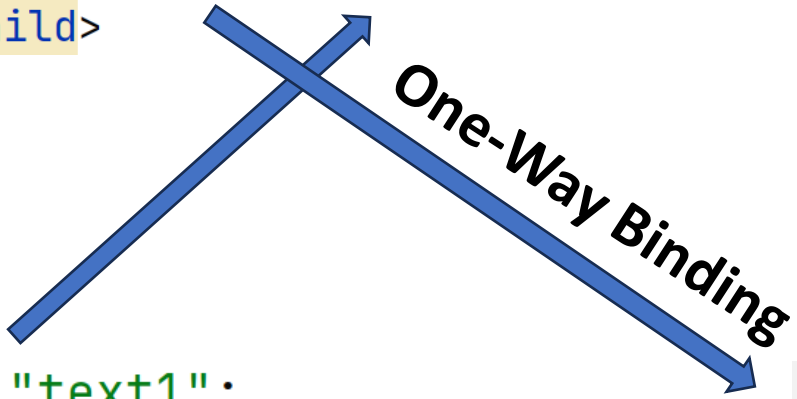
(Reminder) [] = One-Way data binding

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, @for() { }
*ngIf, @if() { }
@Component
[] .. @Input field, input()
() .. @Output changed, output()

[()] .. Bidirectional Binding, model()
router

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

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

```
parentValue: string = "";
```

[()] = 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 ([])

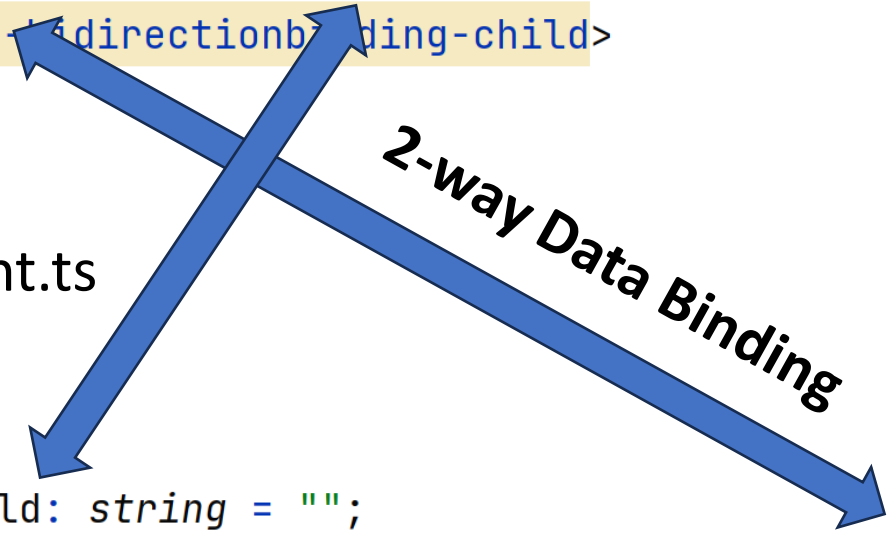
[()] is named = « Banana Box »

… = banana inside a box

[(childField)]=« parentField »

Parent component.html

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



Child component.html

Parent component.ts

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

Child component.ts

```
@Input()
childField: string = "";

@Output()
childFieldChange = new Subject<string>();
```

Angular ≥ 17

field = model<Type>()

test-model.component.ts

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

5+ usages
@Component({
  selector: 'app-test-model',
  standalone: true,
  imports: [],
  templateUrl: './test-model.component.html'
})
export class TestModelComponent {

  field : ModelSignal<string>  = model<string>( initialValue: 'hello');

  1 usage
  onClick() : void  {
    this.field.set(this.field() + 'x');
  }
}
```

[(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 = "";
```

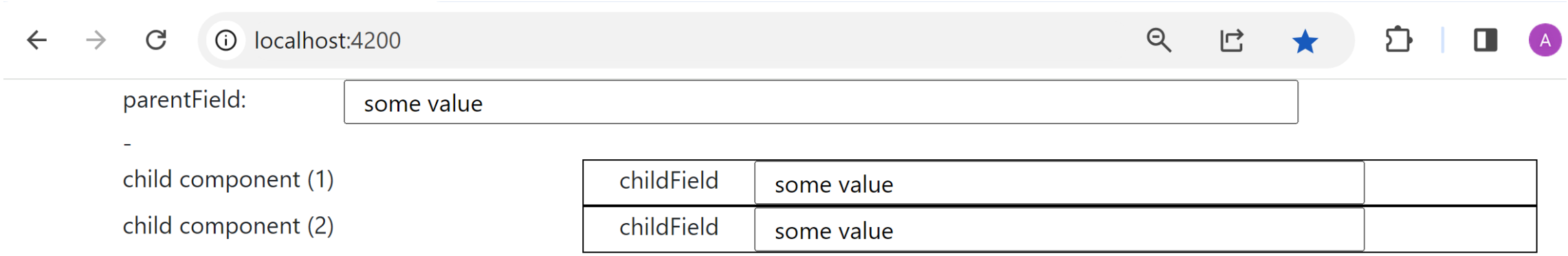
Child .html

```
<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



Outline

templateHtml {{ expr }}
*ngFor, @for() { }
*ngIf, @if() { }
@Component
[] .. @Input field, input()
() .. @Output changed, output()
[()] .. Bidirectional Binding, model()

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.forRoute(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>
```

Angular ≥ 17

providers: [provideRouter(routes),]

app.config.ts

```ts
import {ApplicationConfig, importProvidersFrom} from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import {provideHttpClient} from "@angular/common/http";

2 usages
export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes),
    provideHttpClient(),
    importProvidersFrom(
      // legacy NgModule
    )
  ]
};
```

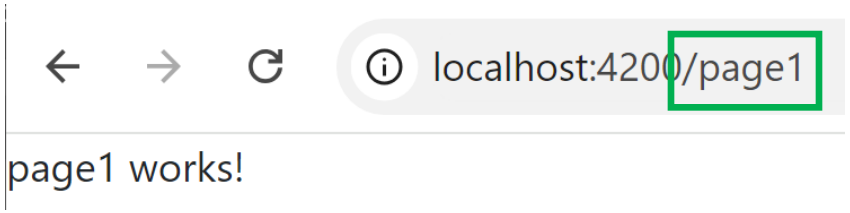
Demo Router

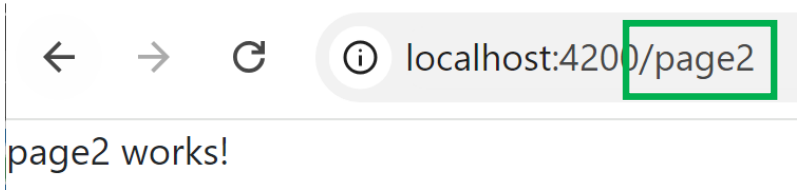
```
$ ng g c demo2/page1
CREATE src/app/demo2/page1/page1.component.html (20 bytes)
CREATE src/app/demo2/page1/page1.component.spec.ts (552 bytes)
CREATE src/app/demo2/page1/page1.component.ts (198 bytes)
CREATE src/app/demo2/page1/page1.component.css (0 bytes)
UPDATE src/app/app.module.ts (792 bytes)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

```
$ 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 },
];
```





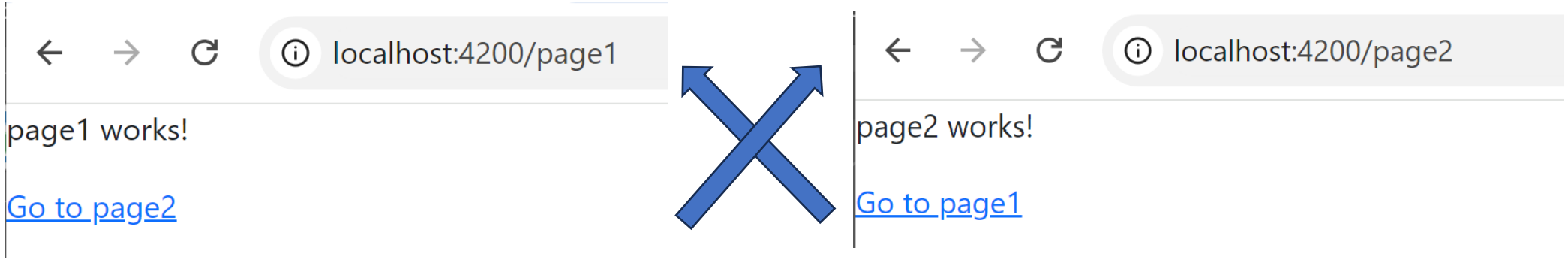
Navigation : <a routerLink=« /page »>

page1.component.html

```
1  <p>page1 works!</p>
2
3  <a routerLink="/page2">Go to page2</a>
```

page2.component.html

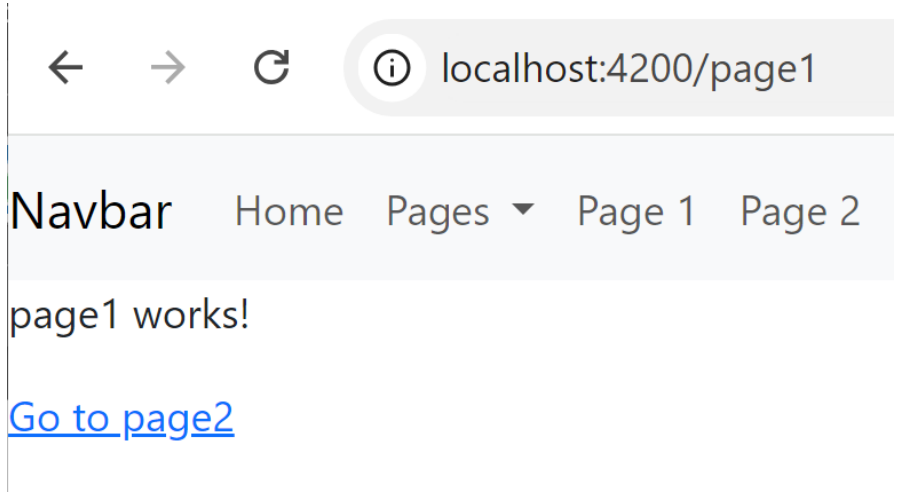
```
1  <p>page2 works!</p>
2
3  <a routerLink="/page1">Go to page1</a>
```

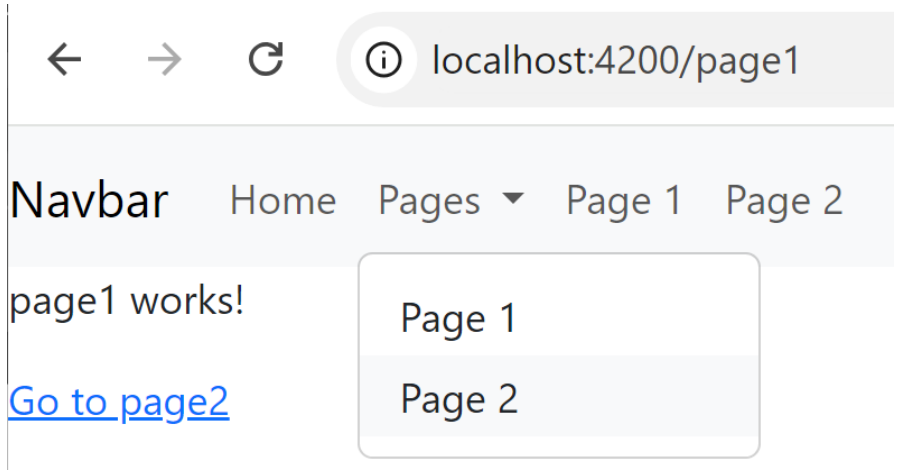


Main page = NavBar menu + <router-outlet> (+ footer)

app.component.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <ul class="navbar-nav mr-auto">
    <li class="nav-item active">
      <a class="nav-link" href="#">Home</a>
    </li>
    <li ngbDropdown class="nav-item">
      <a ngbDropdownToggle id="navbarMenu1" class="nav-link" role="button">
        Pages
      </a>
      <div ngbDropdownMenu aria-labelledby="navbarMenu1" class="dropdown-menu" >
        <a class="dropdown-item" routerLink="/page1">Page 1</a>
        <a class="dropdown-item" routerLink="/page2">Page 2</a>
      </div>
    </li>
    <li class="nav-item">
      <a class="nav-link" routerLink="/page1">Page 1</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" routerLink="/page2">Page 2</a>
    </li>
  </ul>
</nav>

<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)
```

app-store.service.ts

```
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 }
```

Angular ≥ 17

myService = inject(MyService)

todo-list.component.ts

```ts
import {Component, inject} from '@angular/core';
import { NgForOf } from "@angular/common";
import { TodoService } from "../todo.service";

2 usages
@Component({
  selector: 'app-todo-list',
  standalone: true,
  imports: [ NgForOf ],
  templateUrl: './todo-list.component.html',
})
export class TodoListComponent {

  public const todoService : TodoService  = inject(TodoService);

  // constructor(public todoService: TodoService) {
  // }
```

Outline

templateHtml {{ expr }}
*ngFor, @for() { }
*ngIf, @if() { }
@Component
[] .. @Input field, input()
() .. @Output changed, output()
[()] .. Bidirectional Binding, model()
router

Next steps ?

Next steps…