

# Exploring More Lifecycle Hooks

In this lecture, we are going to explore the other lifecycle hooks. Before we begin, I don't recommend stressing over every hook. It's likely you won't need every hook. I'll be sure to point out the most common hooks in Angular applications. The **constructor** and **ngOnInit** hooks are some of the most common hooks you may come across. They're so common that the CLI adds these hooks to a **ng generate component**.

We will start with the **ngOnChanges** and **ngOnCheck** hooks.

First, we should implement their interfaces. We are continuing to work in the **post class**.

At the top of the file, update the import statement for the **OnChanges** and **DoCheck** interfaces.

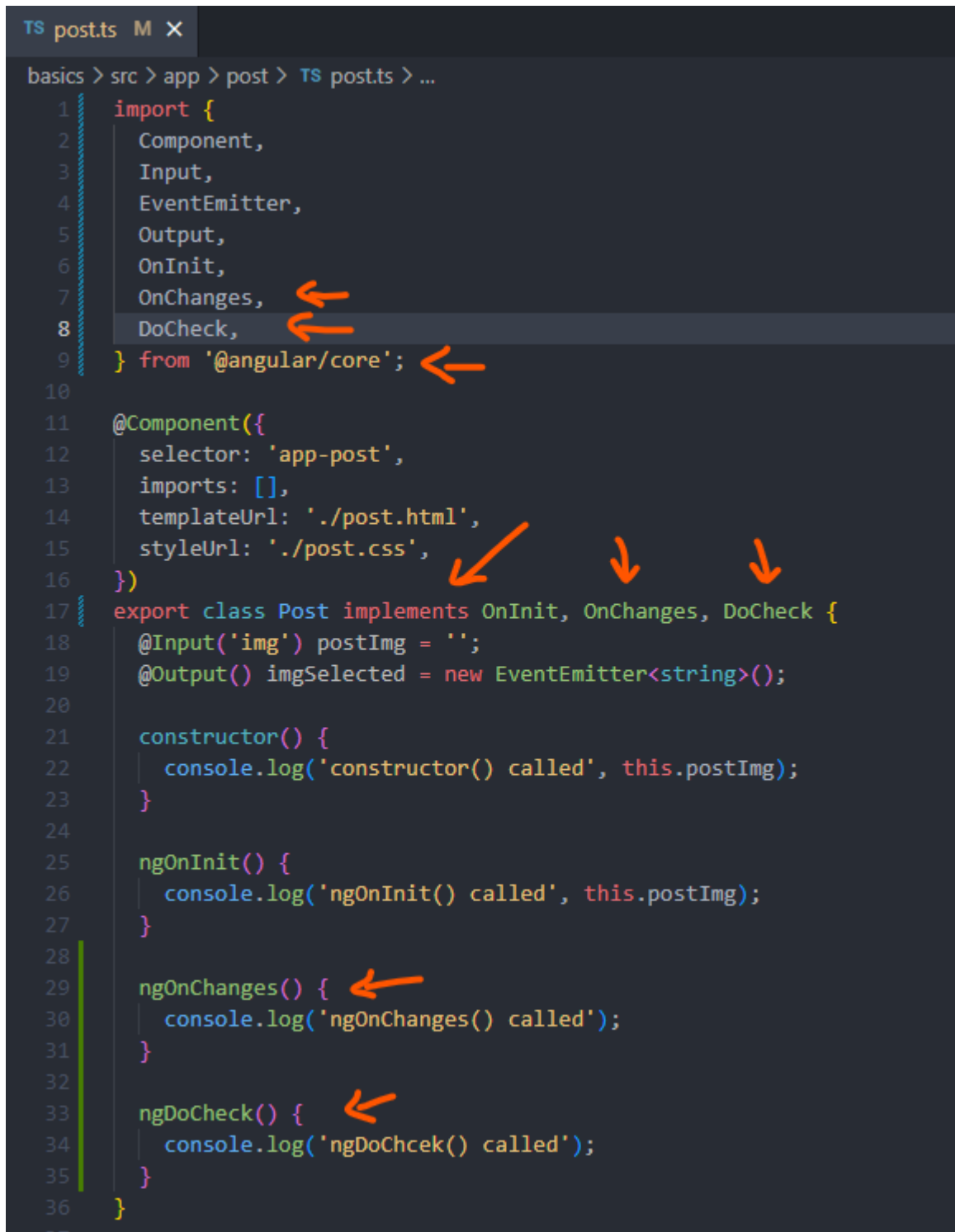
Next, let's implement these interfaces to the class.

Lastly, let's define both functions, the name of the functions is **ngOnChanges()** and **ngDoCheck()**.

Inside both functions, let's log some messages:

```
ngOnChanges(changes: SimpleChanges): void {}  
ngDoCheck(): void {}  
}
```

Automatically  
after defined  
implements



```

1  import {
2    Component,
3    Input,
4    EventEmitter,
5    Output,
6    OnInit,
7    OnChanges,
8    DoCheck,
9  } from '@angular/core';
10
11  @Component({
12    selector: 'app-post',
13    imports: [],
14    templateUrl: './post.html',
15    styleUrls: ['./post.css'],
16  })
17  export class Post implements OnInit, OnChanges, DoCheck {
18    @Input('img') postImg = '';
19    @Output() imgSelected = new EventEmitter<string>();
20
21    constructor() {
22      console.log('constructor() called', this.postImg);
23    }
24
25    ngOnInit() {
26      console.log('ngOnInit() called', this.postImg);
27    }
28
29    ngOnChanges() {
30      console.log('ngOnChanges() called');
31    }
32
33    ngDoCheck() {
34      console.log('ngDoChcek() called');
35    }
36  }

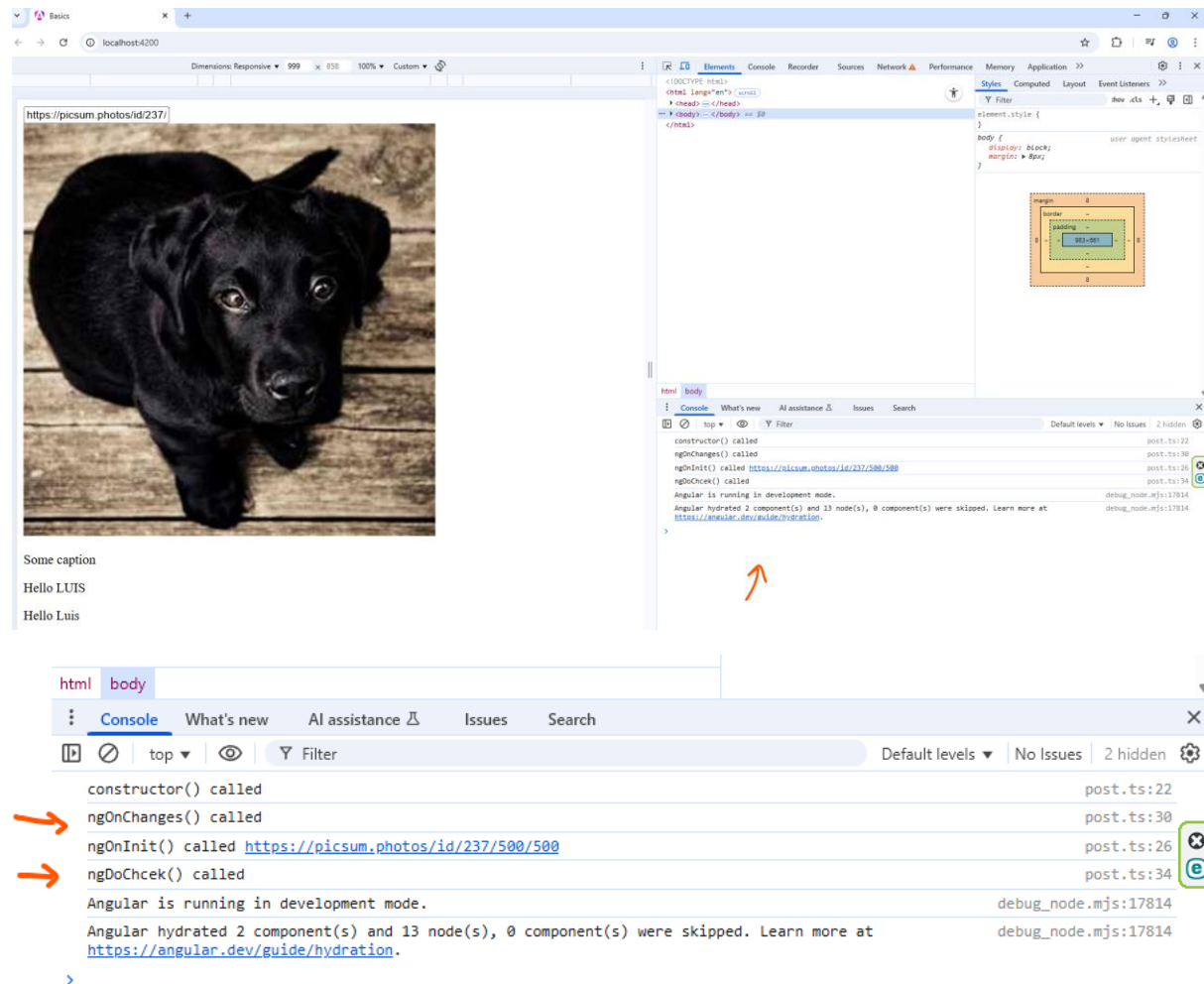
```

Both functions have similar behaviour, unlike the **ngOnInit()** life cycle hook. These two functions will run multiple times for different reasons.

The **ngOnChanges()** function will run whenever changes are made to the component. For example, let's say we update the URL to the image, this function will run whenever that happens. In fact, it's guaranteed to run once in our component. Keep in mind, the post image property is set to an empty string after the data has been passed down from the parent component to the child component, the post image property will be updated. Therefore, Angular will run the **ngOnChange()** function.

The **ngDoChange()** function runs after a change detection cycle has occurred. We've talked about change detection before. It's the system Angular uses for synchronizing a component data with its template. Angular is not picture perfect. It's possible it may overlook an area in our app during change detection. **The purpose of the ngDoCheck function is to perform updates that Angular may miss.** Rarely will you need this function, but it is available whenever Angular fails to update your component properly. The **ngDoCheck** function can be useful for performing changes when Angular doesn't want to.

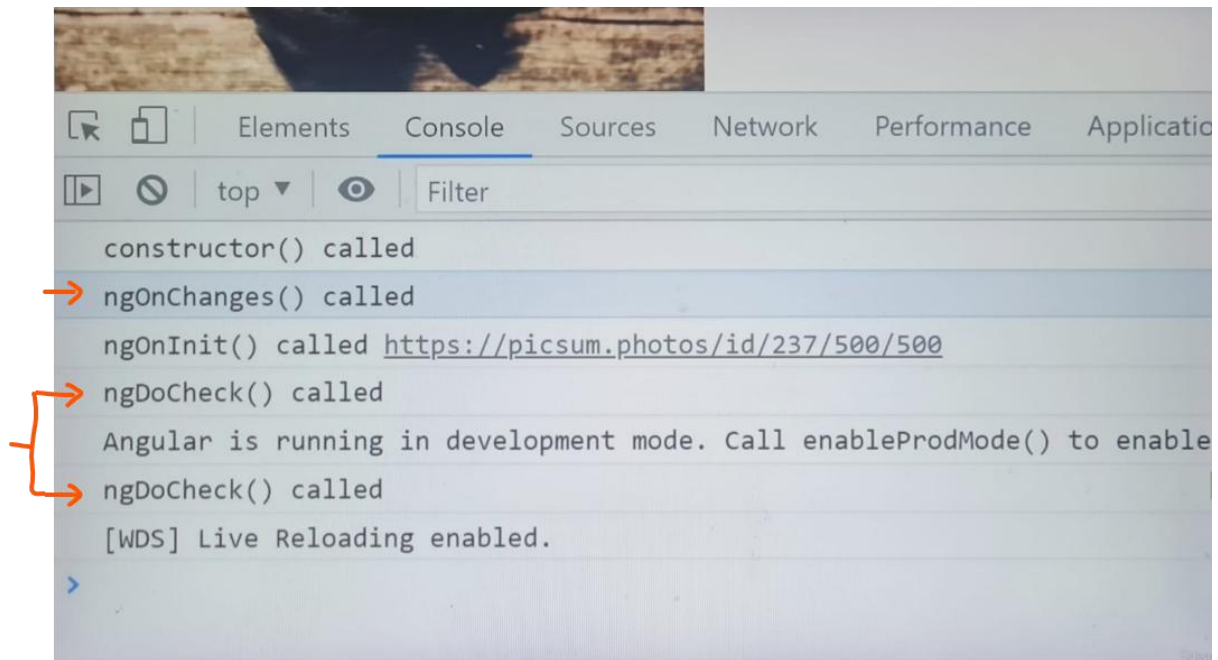
Let's test out both functions in the browser:



The screenshot shows a web browser at localhost:4200 displaying a black puppy image from a Picsum URL. Below the image are three lines of text: "Some caption", "Hello LUIS", and "Hello Luis". The browser's developer tools are open, showing the Console tab with the following log entries:

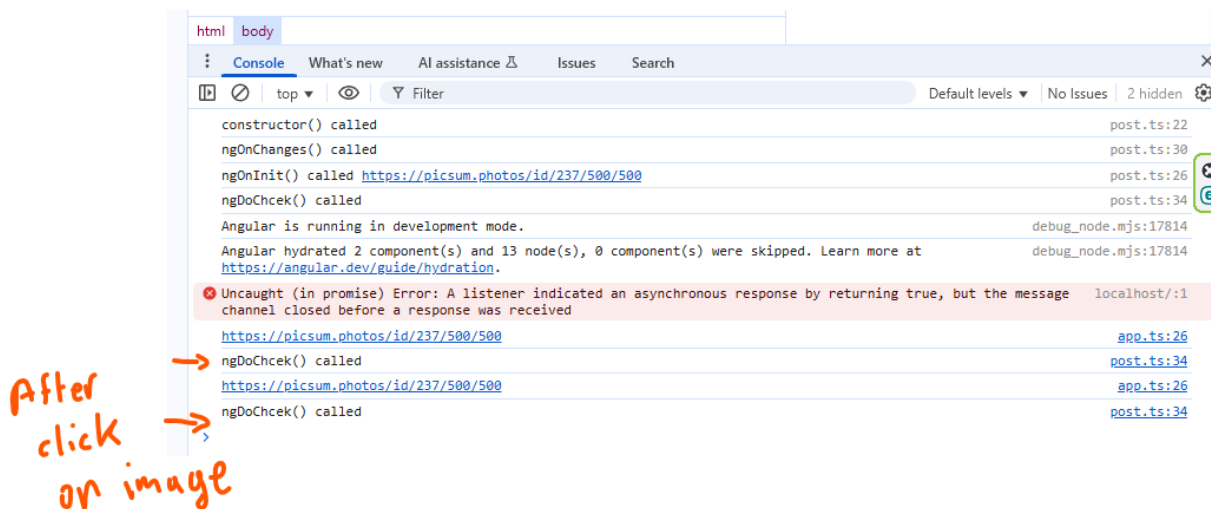
- constructor() called (post.ts:22)
- ngOnChanges() called (post.ts:30)
- ngOnInit() called <https://picsum.photos/id/237/500/500> (post.ts:26)
- ngDoChcek() called (post.ts:34)
- Angular is running in development mode. (debug\_node.mjs:17814)
- Angular hydrated 2 component(s) and 13 node(s), 0 component(s) were skipped. Learn more at <https://angular.dev/guide/hydration>. (debug\_node.mjs:17814)

Orange arrows point to the **ngOnChanges()** and **ngDoChcek()** entries in the console. A green box highlights the **ngDoChcek()** entry, and a red 'X' icon is next to it.



In the console, the **ngOnChanges()** function runs once. Take special note of the order of the messages. This function runs after the **constructor()** function, but before the **ngOnInit()** function, it makes sense, since the **ngOnChanges()** function happens immediately after changes in our components.

The **ngDoCheck()** function runs twice, as a reminder, the change detection system in Angular will run twice to check for errors. Therefore, we can expect this function to run twice. It'll continue to run whenever change detection is triggered. For example, I click on the image, the **ngDoCheck()** function gets called:



Even though our component isn't changing, this function will still run. [**ngDoCheck()**]

There are four more hooks we need to talk about. Before we get into them, we need to distinguish between the view and content in a template.

## A View vs Content

- The view refers to the component's template.
- The content refers to the projected content from the parent component.

In Angular, **content** is placed into two categories, they can either be **view** or **content**.

The **view** refers to the **content** inside the **template file**, whereas, the **content** refers to the **projected content** from the **parent component**.

Angular processes both categories of content separately. To better understand, let's check out the app template file:

```

<> app.html  X
basics > src > app > <> app.html > ...
  Go to component
1  <input (keyup)="changeImage($event)" [value]="imgURL" />
2
3  <app-post [img]="imgURL" (imgSelected)="logImg($event)">
4    <p>Some caption</p>
5  </app-post>
6
7  <p>Hello {{ name.toLocaleUpperCase() }}</p>
8  <p>Hello {{ getName() }}</p>
9  <p>{{ 15 + 13 }}</p>
10
  
```



Inside the `<app-post>` tags, we are inserting content into the component. This process is called **Content Projection**. Even though we're inserting this content into the components. We don't have access to their properties from the post component. For example, the post component has a property called **postImg**. Inside the **projected content**, we can't access this property if we were to write an expression for the **posting** property, we would receive an error:

```

<> app.html 1, M X
basics > src > app > <> app.html > app-post > p
Go to component
1 <input (keyup)="changeImage($event)" [value]="imgURL" />
2
3 <app-post [img]="imgURL" (imgSelected)="logImg($event)">
4   <p>Some caption {{ psotImg }}</p>
5 </app-post>
6
7 <p>Hello {{ name.toLocaleUpperCase() }}</p>
8 <p>Hello {{ getName() }}</p>
9 <p>{{ 15 + 13 }}</p>
10

```

*← Error*

```

NG9: Property 'psotImg' does not exist on type 'App'.
src/app/app.html:4:21

```

Click outside, press **Esc** key, or fix the code to dismiss.

The **postImg** property is inaccessible, since we're in the app components. Angular will process the template in the app component before sending it to the post component. That's important to understand. The post component is not expected to process the content. Expressions or binding in our **Projected Content** will be processed from the parent component.

Next, the template is sent to the child component after it's been sent to the post component. Angular will begin processing the expression and bindings in the post component.

```

<> app.html X
basics > src > app > <> app.html > ...
Go to component
1 <input (keyup)="changeImage($event)" [value]="imgURL" />
2
3 <app-post [img]="imgURL" (imgSelected)="logImg($event)">
4   <p>Some caption</p>
5 </app-post>
6
7 <p>Hello {{ name.toLocaleUpperCase() }}</p>
8 <p>Hello {{ getName() }}</p>
9 <p>{{ 15 + 13 }}</p>
10

```

*←*

These are two separate events.

The projected content is called **content**.

The content in the component template is called **view**.

With that information in mind, there are **four lifecycle hooks**, **two** for the **content** and another **two** for the **view**.

Let's explore all of them.

Switch back to the **post class**.

Let's go to the top of the file, we will need to import their interfaces: **AfterContentInit**, **AfterContentChecked**, **AfterViewInit**, **AfterViewChecked**.

Next, let's apply these interfaces to the class.

Lastly, let's define the functions required by these interfaces, the names of the functions reflect the names of the interfaces, they are prefixed with the letters **ng**, inside each function we will log to identify them in the console:

```

TS post.ts M X
basics > src > app > post > TS post.ts > Post > ngAfterContentChecked
1  import {
2      Component,
3      Input,
4      EventEmitter,
5      Output,
6      OnInit,
7      OnChanges,
8      DoCheck,
9      AfterContentInit,
10     AfterContentChecked,
11     AfterViewInit,
12     AfterViewChecked,
13 } from '@angular/core';
14
15 @Component({
16     selector: 'app-post',
17     imports: [],
18     templateUrl: './post.html',
19     styleUrls: ['./post.css'],
20 })
21 export class Post
22     implements
23         OnInit,
24         OnChanges,
25         DoCheck,
26         AfterContentInit,
27         AfterContentChecked,
28         AfterViewInit,
29         AfterViewChecked
30 {
31     @Input('img') postImg = '';
32     @Output() imgSelected = new EventEmitter<string>();
33
34     constructor() {
35         console.log('constructor() called', this.postImg);
36     }
37

```



```

38   ngOnInit() {
39       console.log('ngOnInit() called', this.postImg);
40   }
41
42   ngOnChanges() {
43       console.log('ngOnChanges() called');
44   }
45
46   ngDoCheck() {
47       console.log('ngDoChcek() called');
48   }
49
50   ngAfterContentChecked(): void {
51       console.log('ngAfterContentChecked() called');
52   }
53
54   ngAfterContentInit(): void {
55       console.log('ngAfterContentInit() called');
56   }
57
58   ngAfterViewChecked(): void {
59       console.log('ngAfterViewChecked() called');
60   }
61
62   ngAfterViewInit(): void {
63       console.log('ngAfterViewInit() called');
64   }
65 }
66

```

**ngAfterContentInit()** - This hook refers to the **projected content** from **parent component**. It runs after the **content has been initialized**. The expressions and bindings have been processed. At this point the **projected content** has been inserted into the child components.

**ngAfterContentChecked()** – This function runs after the **content has been checked for changes**.

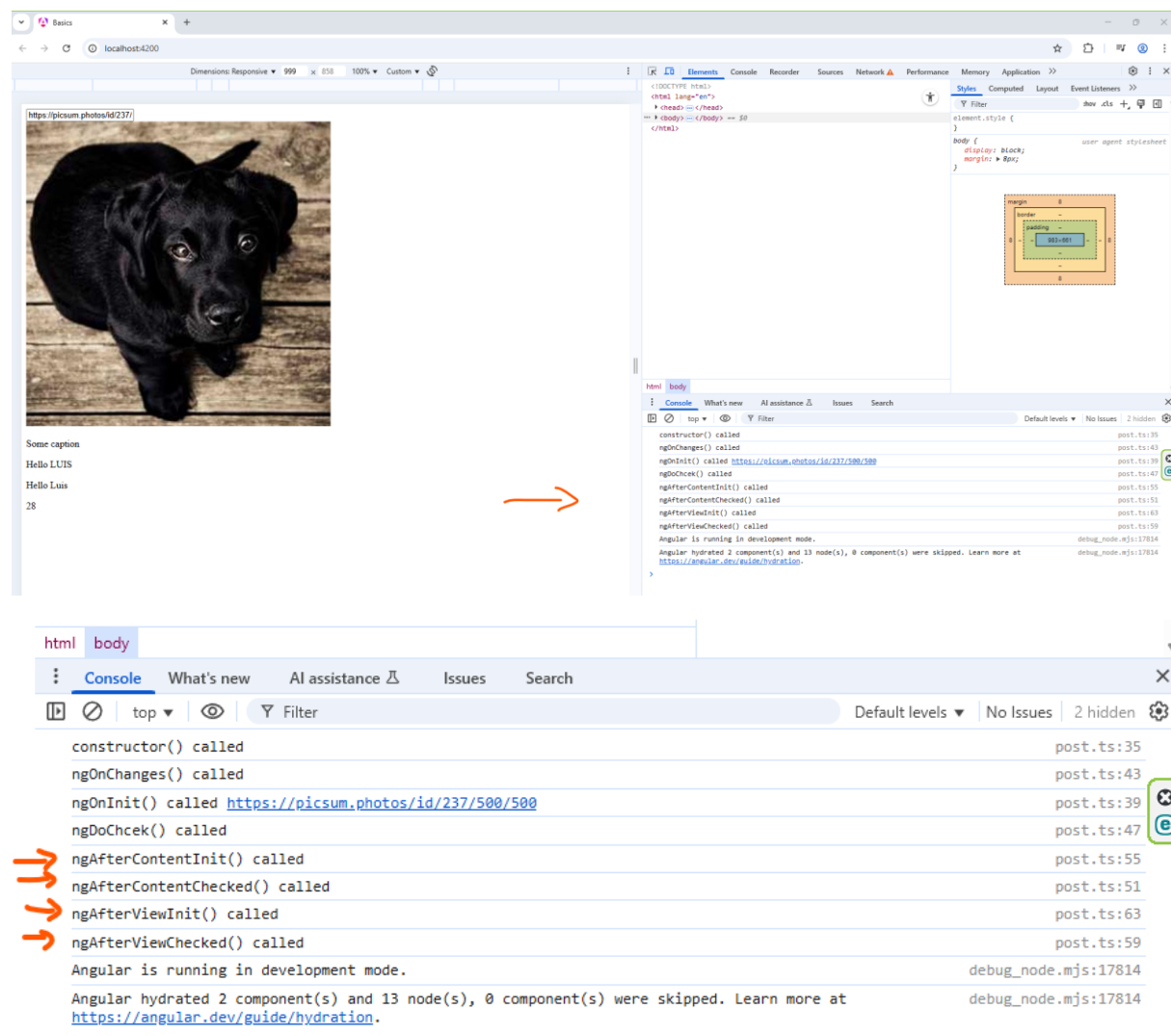
**ngAfterViewInit()** – This function runs when the **components template has been initialized**.

**ngAfterViewChecked()** – This function runs when the **components template has been checked**.

The **viewInit()** and **contentInit()** functions run once. They run before their check function counterparts. It is the first time the content has been processed.

The **contentChecked()** and **viewChceked()** functions can run multiple times. They will be called whenever Angular change detection system runs.

Let's refresh the page in the browser:



In the console, we will find our four messages. Notice how the content function runs before the view.

The **projected content functions** will always run first. These hooks are not commonly used, but they are available if you need them.

There's one last hook we will discuss.

Switch back to the editor.

At the top of the **post class file**, we will update the import list one more time, we will include an interface called **OnDestroy**.

Next, we will implement this interface. Afterward, we will define the **ngOnDestroy()** function in the class.

Lastly, we will log a message:

```
TS post.ts M X
basics > src > app > post > TS post.ts > Post
1  import {
2      Component,
3      Input,
4      EventEmitter,
5      Output,
6      OnInit,
7      OnChanges,
8      DoCheck,
9      AfterContentInit,
10     AfterContentChecked,
11     AfterViewInit,
12     AfterViewChecked,
13     OnDestroy,
14 } from '@angular/core';
15
16 @Component({
17     selector: 'app-post',
18     imports: [],
19     templateUrl: './post.html',
20     styleUrls: ['./post.css'],
21 })
22 export class Post
23     implements
24         OnInit,
25         OnChanges,
26         DoCheck,
27         AfterContentInit,
28         AfterContentChecked,
29         AfterViewInit,
30         AfterViewChecked,
31         OnDestroy
32 {
33     @Input('img') postImg = '';
34     @Output() imgSelected = new EventEmitter<string>();
35
36     constructor() {
37         console.log('constructor() called', this.postImg);
38     }
39 }
```

```
39
40   ngOnInit() {
41     console.log('ngOnInit() called', this.postImg);
42   }
43
44   ngOnChanges() {
45     console.log('ngOnChanges() called');
46   }
47
48   ngDoCheck() {
49     console.log('ngDoChcek() called');
50   }
51
52   ngAfterContentChecked(): void {
53     console.log('ngAfterContentChecked() called');
54   }
55
56   ngAfterContentInit(): void {
57     console.log('ngAfterContentInit() called');
58   }
59
60   ngAfterViewChecked(): void {
61     console.log('ngAfterViewChecked() called');
62   }
63
64   ngAfterViewInit(): void {
65     console.log('ngAfterViewInit() called');
66   }
67
68   ngOnDestroy(): void {
69     console.log('ngOnDestroy() called');
70   }
71 }
72
```

A hand-drawn orange bracket is placed to the right of the `ngOnDestroy()` method definition, spanning from line 68 to line 70. An orange arrow points from the right towards the bracket.

The **OnDestroy Hook** will run when the component is destroyed. We can remove components from the page when we don't need them anymore. If the component is being removed from the document, Angular will run this hook. It'll give us the opportunity to remove functionality to prevent memory leaks.

Unfortunately, we won't have a chance to test this function. We haven't learned how to destroy components yet.

The most commonly used hooks are the:

## Commonly used hooks

- `constructor()`
- `ngOnInit()`
- `ngOnChanges()`
- `ngOnDestroy()`

Before we move on, there's one last warning I want to give. **Hooks that run multiple times can impact the performance of your app. If you need to use these hooks, make sure you're not performing an intensive action. Otherwise, the performance of your app may suffer.**

## Be careful

Hooks that run more than once  
can impact the performance of  
your app.