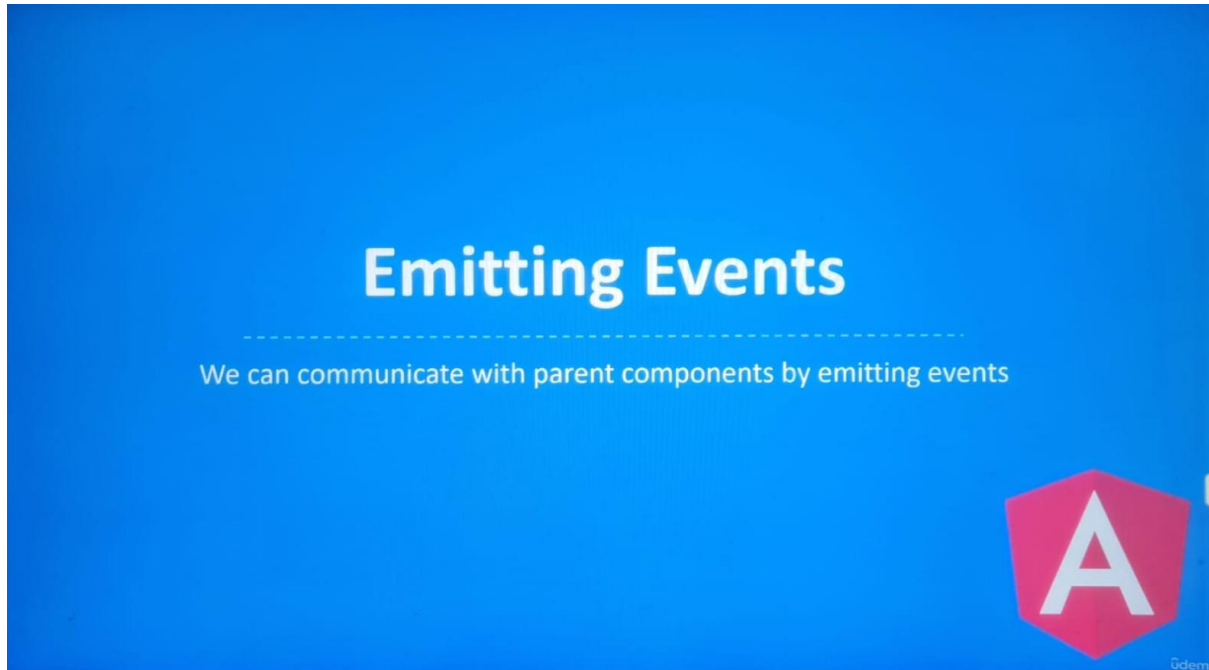


# Emitting Events

In this lecture, we're going to explore communication between components with a feature called outputs.

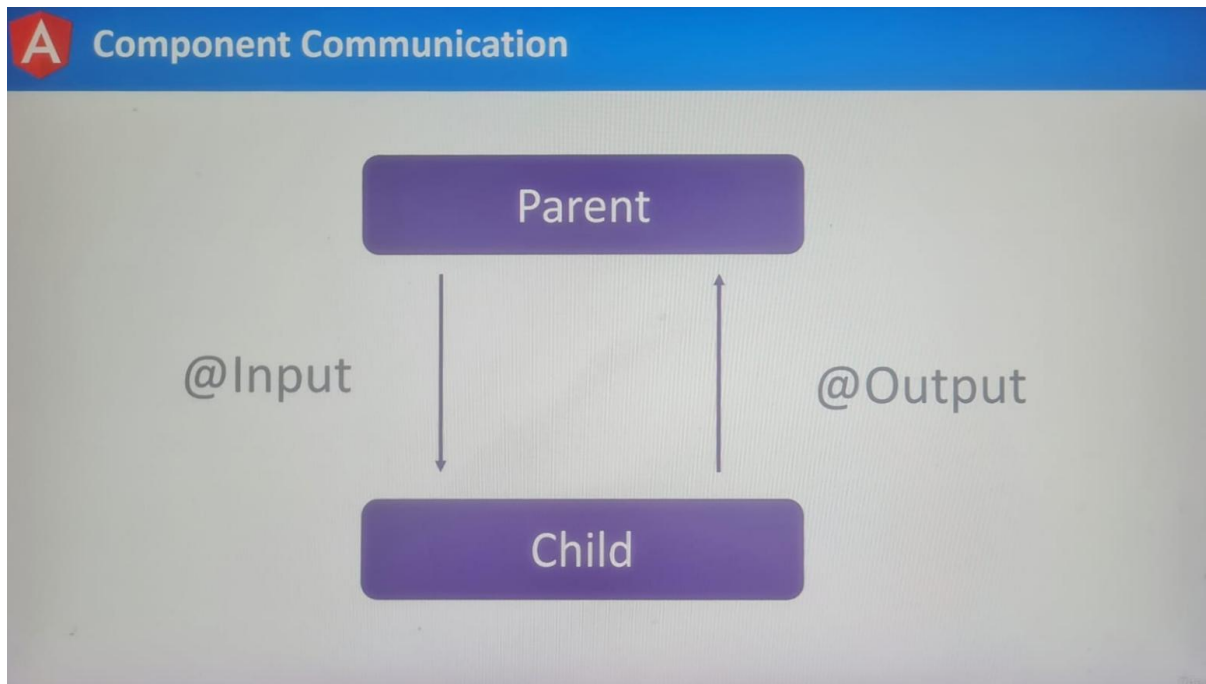


At the moment, the communication between parent and child components is one way.

Parent component can send data to child component with `@Inputs()`.

What if we want to send data from a child component to a parent component? If we want to communicate with parent components, we can emit **custom events**.

Communicating from a child component to a parent component is similar to parent to child communication. We can send data to a child component by using the input decorator. Similarly, we can send data back up to a parent component by adding the output decorator:



There are a couple of steps we need to take, but overall, the process is still the same.

For this demonstration, let's inform the parent component if the image was selected, we will be working inside the post component.

```

1  import { Component, Input } from '@angular/core';
2
3  @Component({
4    selector: 'app-post',
5    templateUrl: './post.component.html',
6    styleUrls: ['./post.component.css']
7  })
8  export class PostComponent {
9    @Input('img') postImg = ''
10  }
11

```

File: src/app/post/post.component.ts

The first step to communicating with a parent component is to **emit an event**. It's easy to work with **browser events**. The browser will **emit** them on our behalf. All we have to do is listen for them.

**Custom events** are another story.

We need to create a system for **emitting and managing events**. Luckily, we don't have to. Angular has a class for creating and managing **custom events**, at the top of the file, we're going to update the

import statement from the **@angular/core** package to include the **EventEmitter** class [Emit = לפלוט]. This object will help us with creating events.

Let's create a new property in our class called **imgSelected**.

Normally, event should be prefixed with the word **on**, however, Angular advises against that.

<https://v17.angular.io/guide/styleguide#dont-prefix-output-properties>

**Don't prefix *output* properties.**

The value for this property **imgSelected** will be a new instance of the **EventEmitter** class.

By creating a new instance of this class, we've made a **custom event**, through this instance, we can emit the event whenever we want. Before we do, we should add types safely to our events. We have complete control over the **data emitted by our events**. At the moment, the data can be anything, we can send numbers, strings, objects, etc.

```

src > app > post > post.component.ts > PostComponent > imgSelected
1  import { Component, Input, EventEmitter } from '@angular/core';
2
3  @Component({
4    selector: 'app-post',
5    templateUrl: './post.component.html',
6    styleUrls: ['./post.component.css']
7  })
8  export class PostComponent {
9    @Input('img') postImg = ''
10   imgSelected = new EventEmitter()
11 }
12

```

If we hover our mouse over the object, it'll tell us the type of data emitted from this event:

```

TS posts M x
basics > src > app > post > TS pos
1  import { Component,
2
3  @Component({
4    selector: 'app-pos
5    imports: [],
6    templateUrl: './po
7    styleUrls: ['./post.
8  })
9  export class Post {
10   @Input('img') post ({
11   imgSelected = new EventEmitter();
12 }
13

```

(alias) new EventEmitter(isAsync?: boolean): EventEmitter<any> (+1 overload)  
import EventEmitter

Use in components with the **@Output** directive to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance.

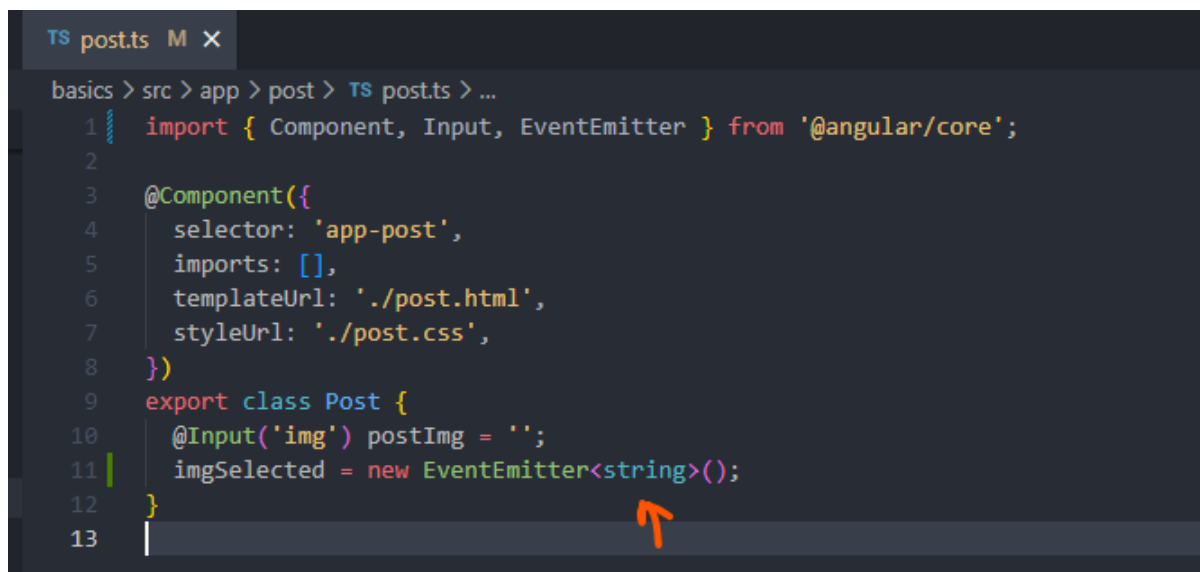
**@usageNotes**  
Extends **RxJS Subject** for Angular by adding the **emit()** method.

In the following example, a component defines two output properties that create event emitters. When the title is clicked, the emitter emits an open or close event to toggle the current visibility state.

The data type is set to **any**, the **any** type is something we should avoid unless we truly want to **emit any** type of value. For this example, we're going to send the parent component, the URL of the clicked image, the URL is a string, we should add type safety to the values we emit. We can add type

safety by using **generics**. After the name of the class, we will add a generic, the type will be set to string:

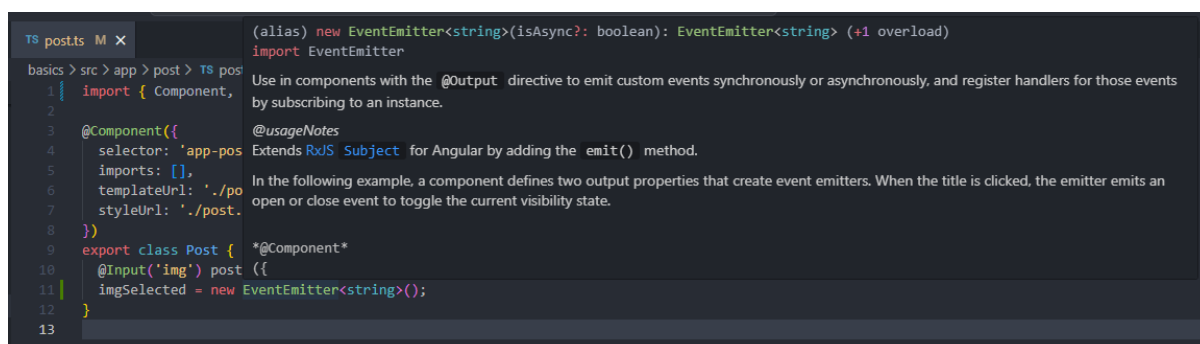
```
imgSelected = new EventEmitter<string>();
```



```

TS post.ts M X
basics > src > app > post > TS post.ts > ...
1  import { Component, Input, EventEmitter } from '@angular/core';
2
3  @Component({
4    selector: 'app-post',
5    imports: [],
6    templateUrl: './post.html',
7    styleUrls: ['./post.css'],
8  })
9  export class Post {
10    @Input('img') postImg = '';
11    imgSelected = new EventEmitter<string>();
12  }
13
  
```

Let's hover our mouse over the **EventEmitter** object again, this time the type has been set to string:



```

TS post.ts M X
basics > src > app > post > TS post
1  import { Component,
2
3  @Component({
4    selector: 'app-pos
5    imports: [],
6    templateUrl: './po
7    styleUrls: ['./post.
8  })
9  export class Post {
10    @Input('img') post ({
11    imgSelected = new EventEmitter<string>();
12  }
13
  (alias) new EventEmitter<string>(isAsync?: boolean): EventEmitter<string> (+1 overload)
  import EventEmitter
  Use in components with the @Output directive to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance.
  @usageNotes
  Extends RxJS Subject for Angular by adding the emit() method.
  In the following example, a component defines two output properties that create event emitters. When the title is clicked, the emitter emits an open or close event to toggle the current visibility state.
  *@Component*
  
```

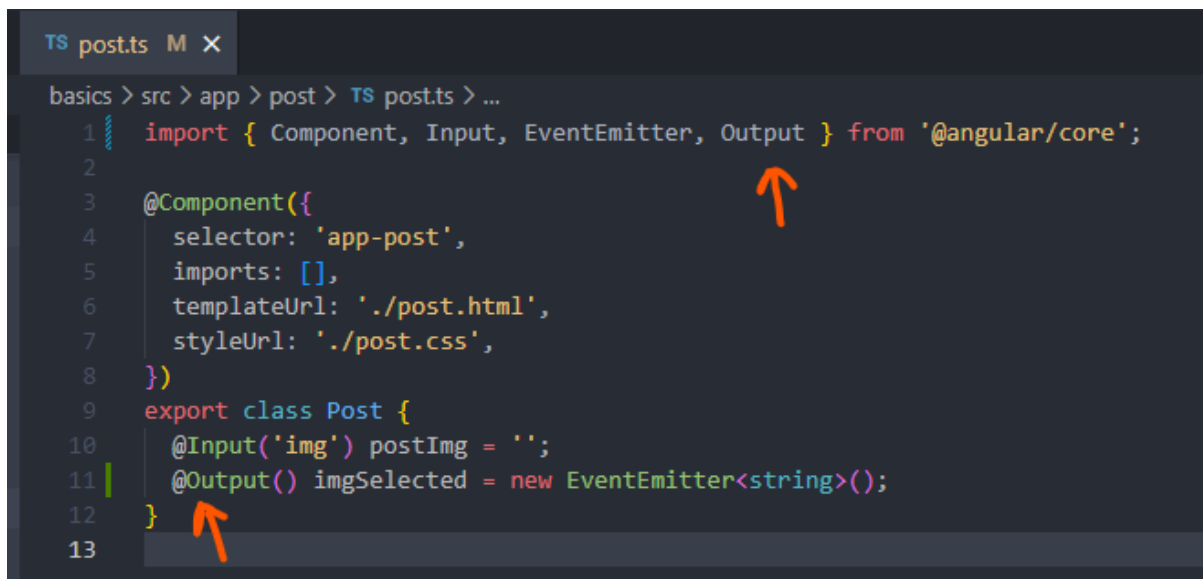
We've successfully added type safety to our **custom event**.

**Events** are the primary way of communicating from a child component to a parent component, which makes sense.

That's how HTML elements communicate data with us. Angular simulates this behaviour for our components. There's one problem with our **custom event**. The event we created is only accessible to the component itself.

**Parent components cannot listen to this event. It's the same reason as before. Angular likes to isolate our components. We don't want external code to interact with the properties from within the class unless we give permission. In our case, we do want to allow the parent component to listen for this event. We can instruct Angular to give permission to parent components by adding the @Output() decorator on the @angular/core package.**

We're going to update the list of imports by adding the **Output decorator** function:



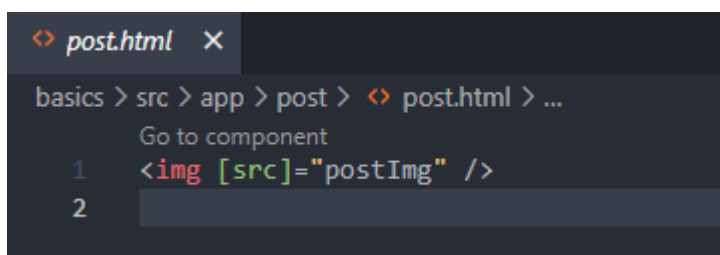
```

TS post.ts M X
basics > src > app > post > TS post.ts > ...
1 import { Component, Input, EventEmitter, Output } from '@angular/core';
2
3 @Component({
4   selector: 'app-post',
5   imports: [],
6   templateUrl: './post.html',
7   styleUrls: ['./post.css'],
8 })
9 export class Post {
10   @Input('img') postImg = '';
11   @Output() imgSelected = new EventEmitter<string>();
12 }
13

```

There are two more steps we need to take before we're finished, the next step is to **emit our event**. At the moment, we've created it, but it will not automatically get **emitted**.

Let's try to handle this part of the process, open the **post component template**:



```

<> post.html X
basics > src > app > post > <> post.html > ...
Go to component
1 <img [src]="postImg" />
2

```

On the image tag, we're going to listen for the **click event**, remember to wrap the **event** with parentheses (), otherwise, Angular will interpret this event as a regular HTML attribute.

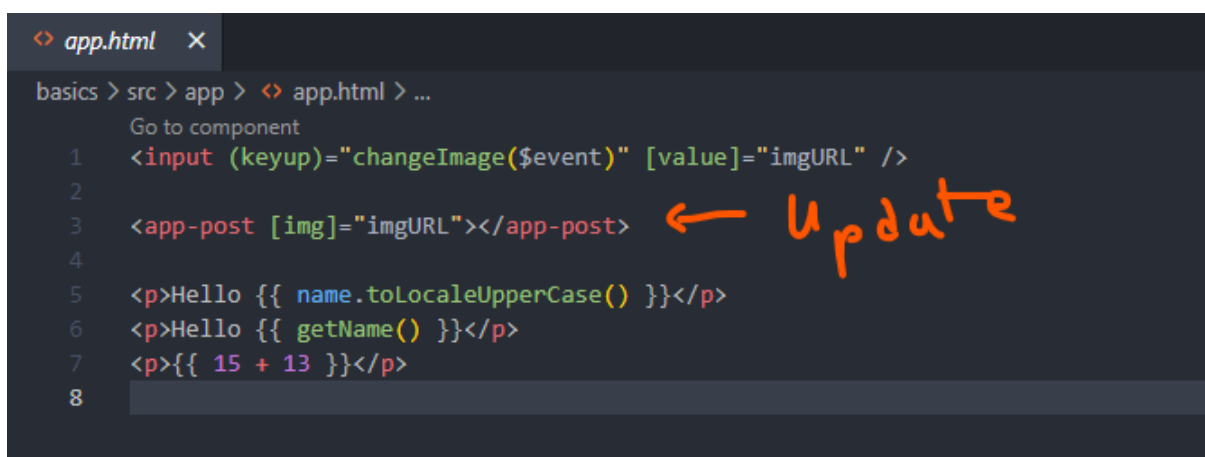
Next, inside the value, we're going to call the **imgSelected.emit()** function on the imgSelected object. The **EventEmitter object** has a **method for emitting an event called emit()**, by calling it, we're triggering the event, we can send data to **event listeners by passing in the data to the emit method**.

Let's send the **postImg** property:



The post component is ready. The last step is to **listen for the event in the parent component**.


Let's switch over the app template file:



On the **app-post** component, we're going to add a pair of parentheses () for our **custom event** – **imgSelected**. Next, we will run the method called **logImg()**. The **logImg()** method will log the image sent by the component. This method doesn't exist in our class, we should define it, before we do, we should provide this image with the **data emitted by our event**. **Data emitted by an Event is stored in the event variable**:

**(imgSelected)="logImg(\$event)"**

```
<> app.html 1, M 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)"></app-post>
4
5  <p>Hello {{ name.toLocaleUpperCase() }}</p>
6  <p>Hello {{ getName() }}</p>
7  <p>{{ 15 + 13 }}</p>
8
```



Let's move on to defining this method, open the app component class:

We will define the `logImg()` method with the **event object as an argument, the type for the event object will be a string.**

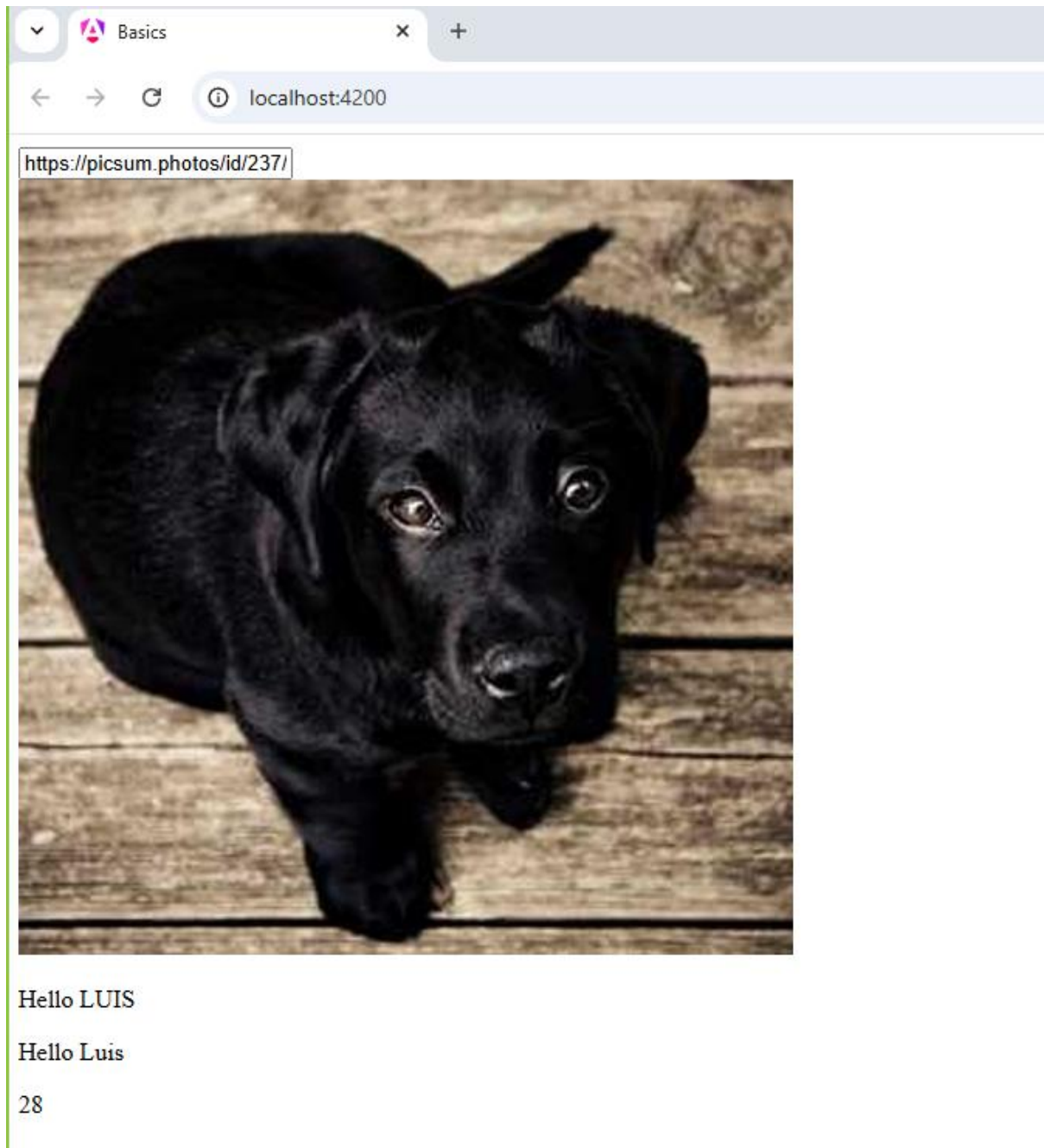
**The value emitted from our event is a string. Therefore, we don't need to set this argument to an event object like we did last time with our changed text method.**

Lastly, we will log the event object:

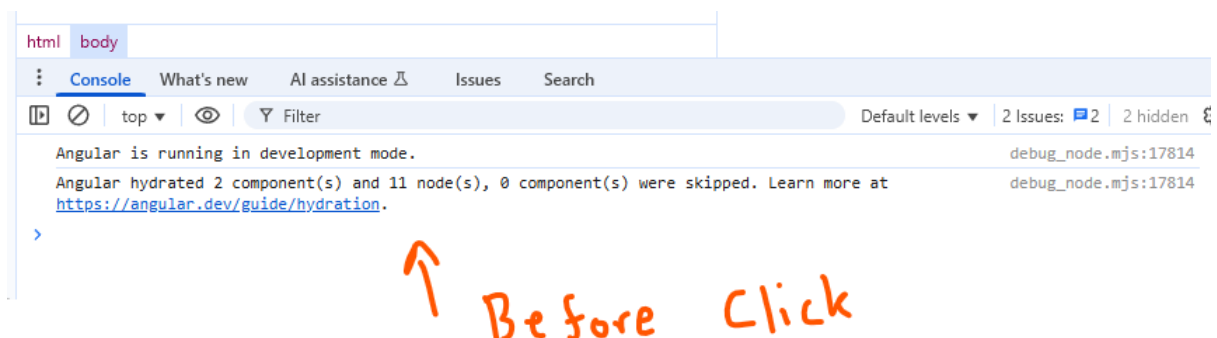
```
TS app.ts 1, M X
basics > src > app > TS app.ts > App > logImg
1  import { Component } from '@angular/core';
2  import { RouterOutlet } from '@angular/router';
3  import { Post } from './post/post';
4
5  @Component({
6    selector: 'app-root',
7    imports: [RouterOutlet, Post],
8    templateUrl: './app.html',
9    styleUrls: ['./app.css'],
10 })
11 export class App {
12   protected title = 'basics';
13
14   protected name = 'Luis';
15   protected imgURL = 'https://picsum.photos/id/237/500/500';
16
17   getName() {
18     return this.name;
19   }
20
21   changeImage(e: KeyboardEvent) {
22     this.imgURL = (e.target as HTMLInputElement).value;
23   }
24
25   logImg(event: string) {
26     console.log(event);
27   }
28 }
29
```

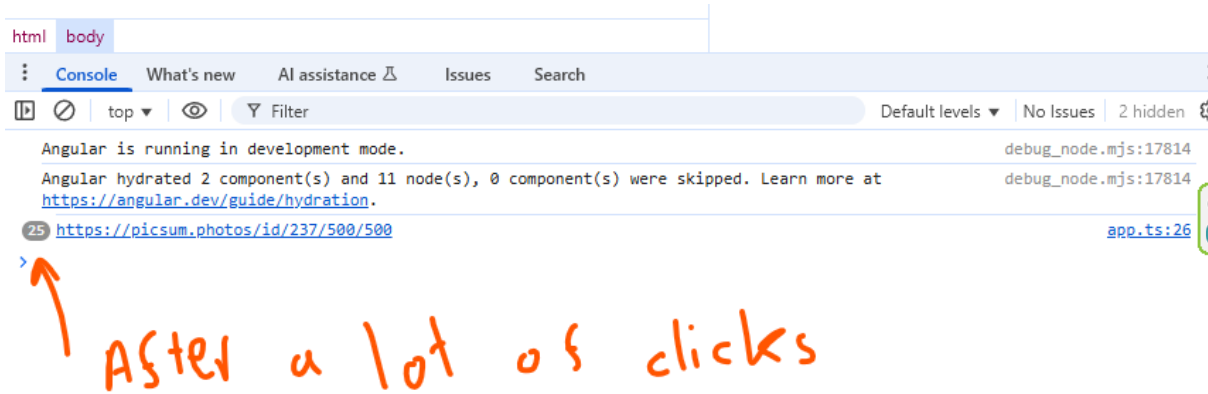
Let's refresh the page in the app:





The app is still working, if we open the developers tools, there shouldn't be errors in the console, let's try clicking on the image:





As we do, the URL of the image is being logged, it's not very exciting. All we're doing is sending the data back and forth between the parent and child components. However, it does show how we can communicate data between components. So you may have noticed the value of the event arguments. Unlike last time, the event arguments it's not an object with dozens of properties about the event. It doesn't include information like the name of the element we clicked on. Whenever we're creating **custom events**, we have complete control over the event arguments, we can choose to send simple data, it depends on the needs of your event. Keep that in mind when working with **custom events**.

**In the next lecture, we're going to look at an alternative option for passing down content from the parent component to the child component.**