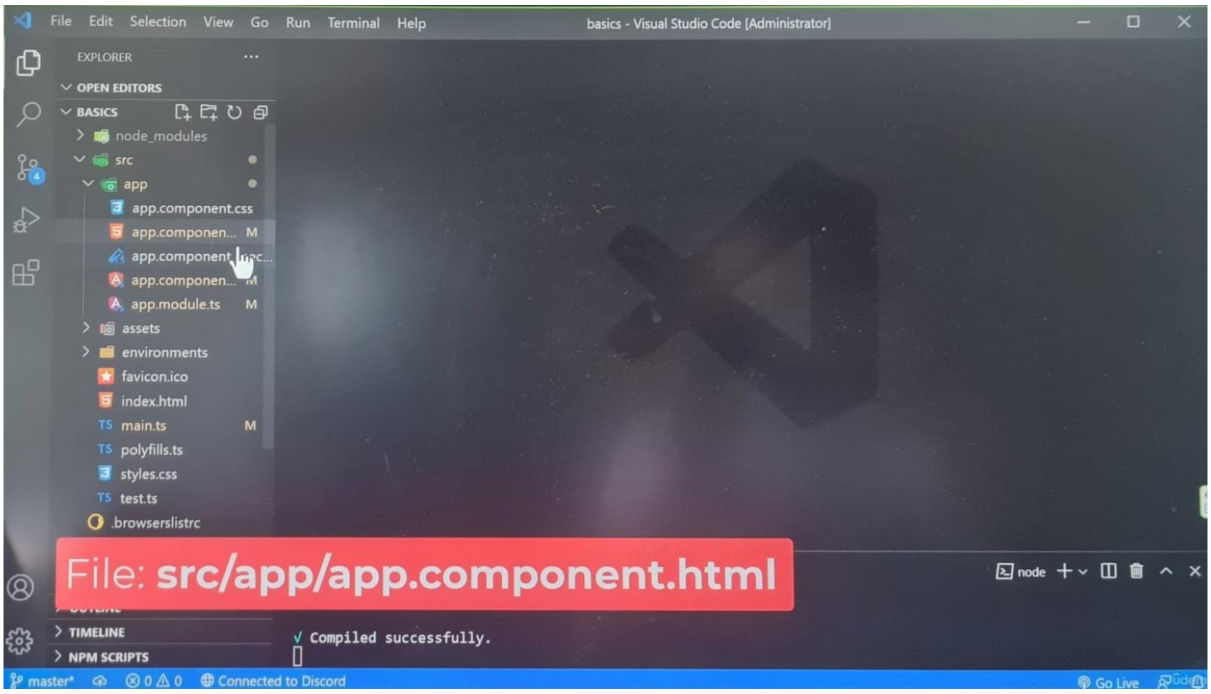
Event Binding

In this lecture, we're going to learn how to listen for events in our templates. At the moment the image will always remain the same, I think it would be cool if we could dynamically swamp this image with another image.

Let's try giving it a shot.



File: src/app/app.component.html

```
√ Compiled successfully.
```

Normally, we would listen for an event by selecting the element with a query function, afterwards we would attach an **Event Listener** to the elements.

Angular provides an easier way to attach **Event Listeners on Elements**, Actually, I would even go as far to say, it's better than Vanilla JavaScript. The syntax is easier to read and write. **All browser Events are supported in Angular, we can handle 'clicks', 'form submissions', 'keyboard events' etc.**

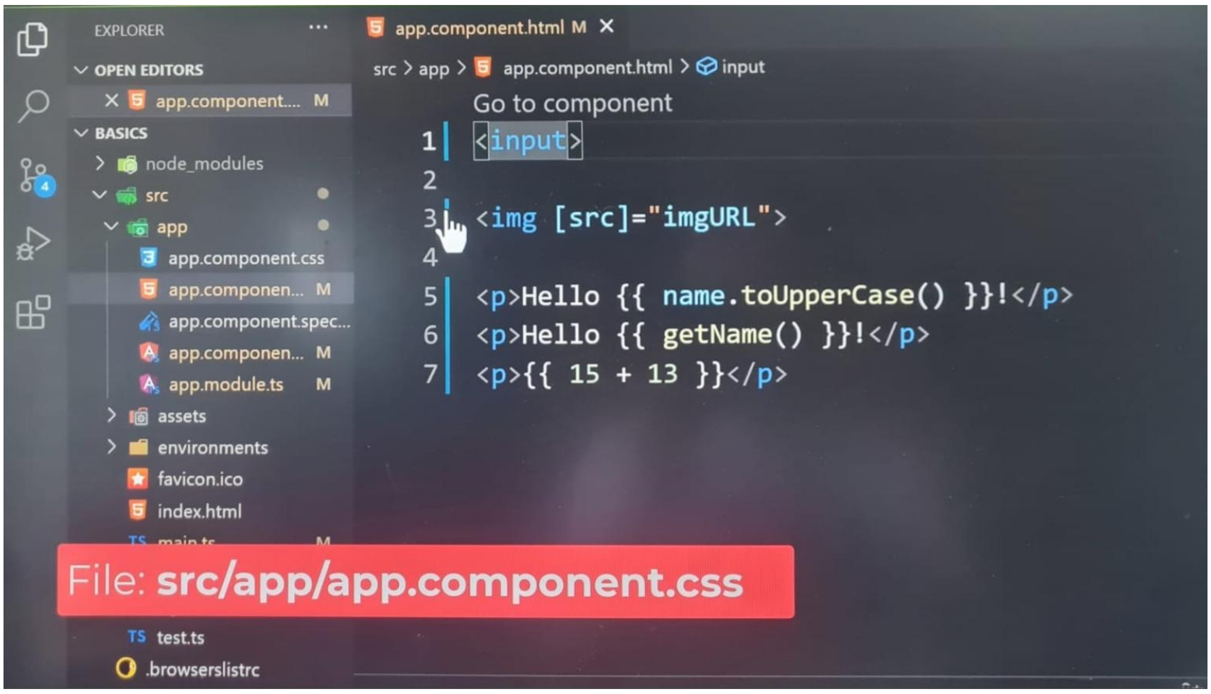
Custom Events are supported too.

We'll look at how to create **Custom Events** in a future lecture.

For now, we're going to focus on **Browser Events**.

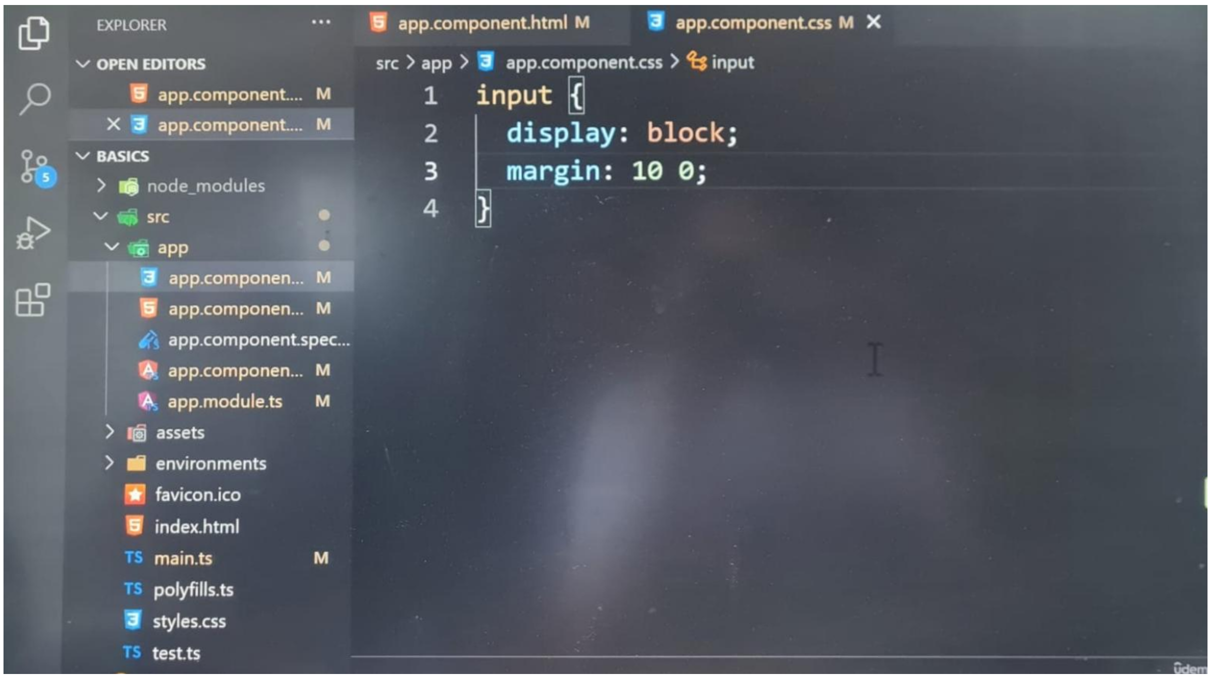
Above the image, we're going to add an input element, through this element, we will allow users to change the image.

Before we move forward, let's add some spacing, open the app stylesheet file:



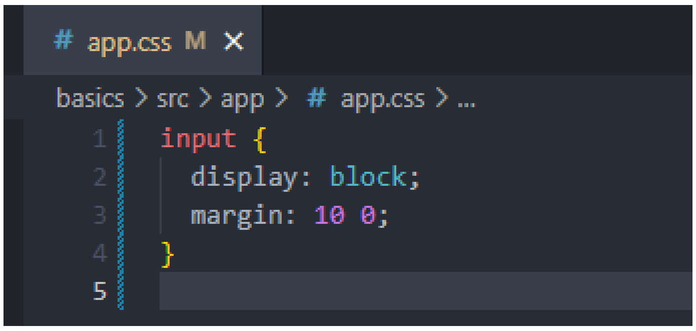
File: src/app/app.component.css

```html
<input>

<img [src]="imgURL">

<p>Hello {{ name.toUpperCase() }}!</p>
<p>Hello {{ getName() }}!</p>
<p>{{ 15 + 13 }}</p>
```



```css
input {
  display: block;
  margin: 10 0;
}
```

On my latest Angular version: [app.css]



```css
input {
  display: block;
  margin: 10 0;
}
```

Let’s go back to the templates.

On the input element, **we are going to listen for an Event called keyup.**

Events can be written like attributes on an element, we need to wrap the event with a pair of parentheses (), if we omitted the parentheses, Angular would interpret this event as a **regular attribute. It’s important to wrap the event with parentheses.**

At this point, you may have started to notice a pattern, HTML template files are not regular HTML files, Angular enhances templates with additional syntax for interacting with the documents. This set of features are why developers love Angular. Our components, template and logic are separated at the same time, they’re not entirely decoupled from one another, we can apply special syntax to our templates to bind properties and events. Without Angular, the browser wouldn’t know how to handle this syntax.

Normally, events are prefixed with the word **on**. We can safely omit the **on** keyword from the event name. This applies to all browser events. Angular automatically attaches this keyword to your **event bindings**. The **keyup event** will fire after a key has been released. We can use this event to help us detect changes in the input. The value of this event can be an expression. We’re going to instruct Angular to run a method. **We will run a method called changeImage() with the event object passed in**.

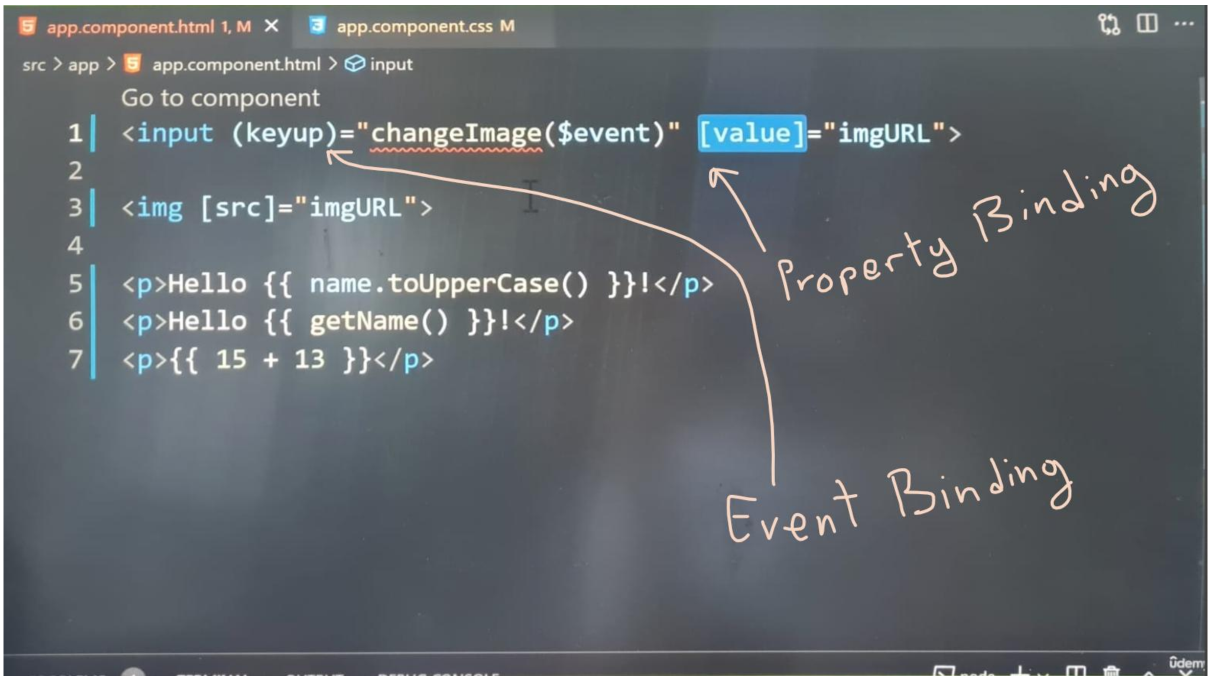
On the one hand, we could handle the logic in the template, however, that’s not recommended. We should outsource complex logic into the class. The template should listen to events, afterward it can pass on **event handling** to our class. Angular will provide us with an object called **event** in the template. We can pass it on to our components methods.

Let’s help the user by providing the initial URL to the image, on the input element, we’re going to **bind** the value attribute to the image URL property. Be sure to wrap the value attribute with square brackets [], otherwise, Angular will not be able to process the expression in the attributes value.

There is a distinction between **binding events and properties**.

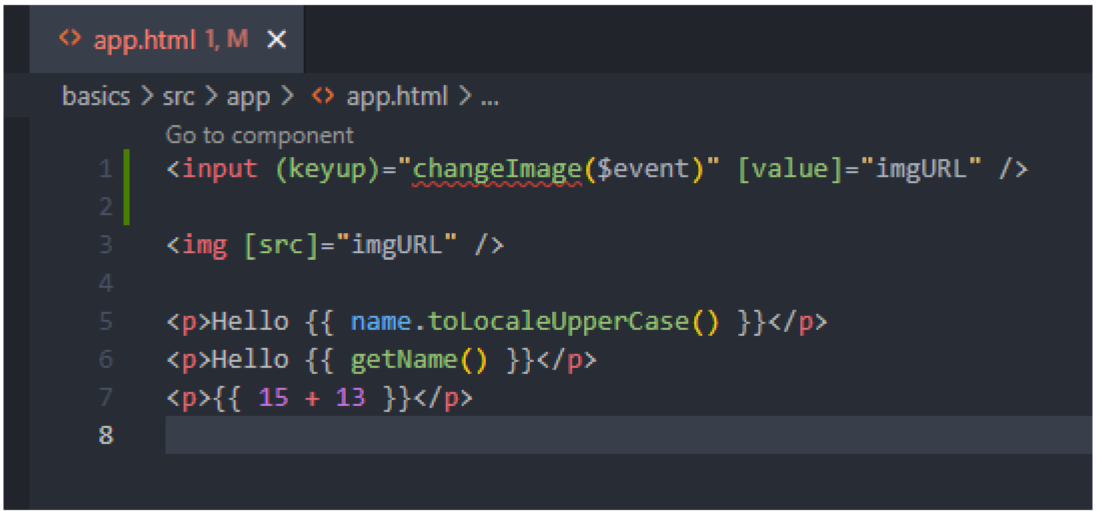
Property binding is performed with square brackets [].

Event binding is performed with parentheses ().



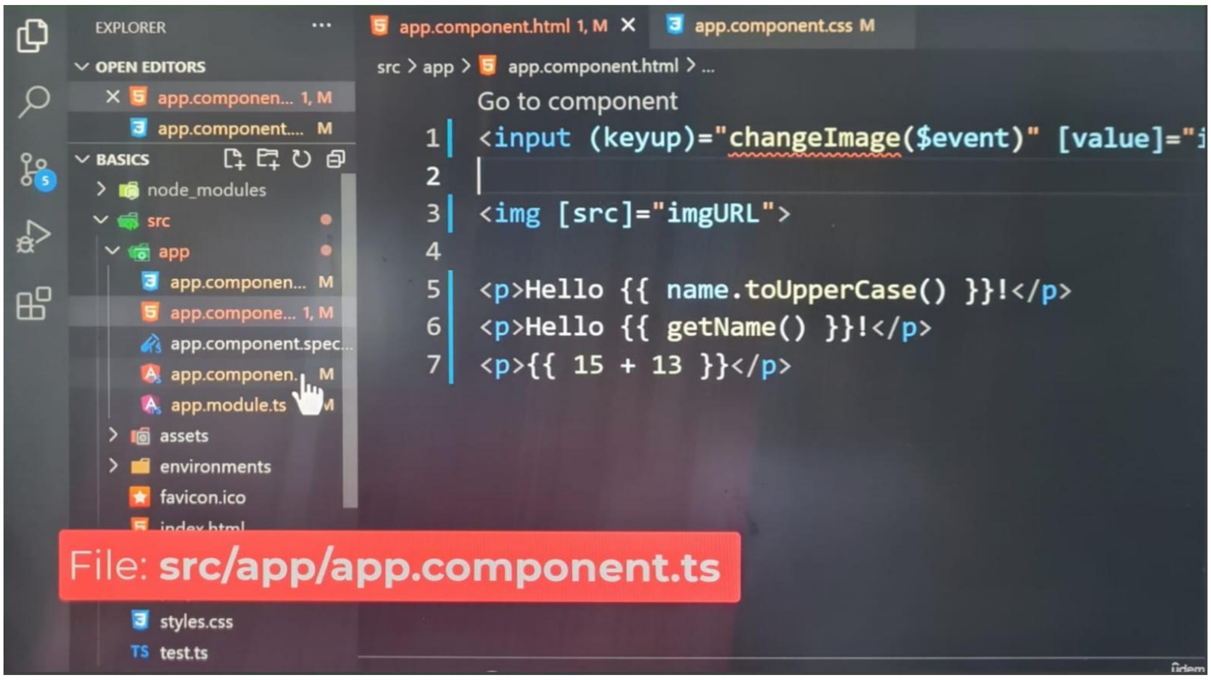
```
<input (keyup)="changeImage($event)" [value]="imgURL">

<img [src]="imgURL">

<p>Hello {{ name.toUpperCase() }}!</p>
<p>Hello {{ getName() }}!</p>
<p>{{ 15 + 13 }}</p>
```

On my PC with latest version of Angular 20.0.5 – app.html:



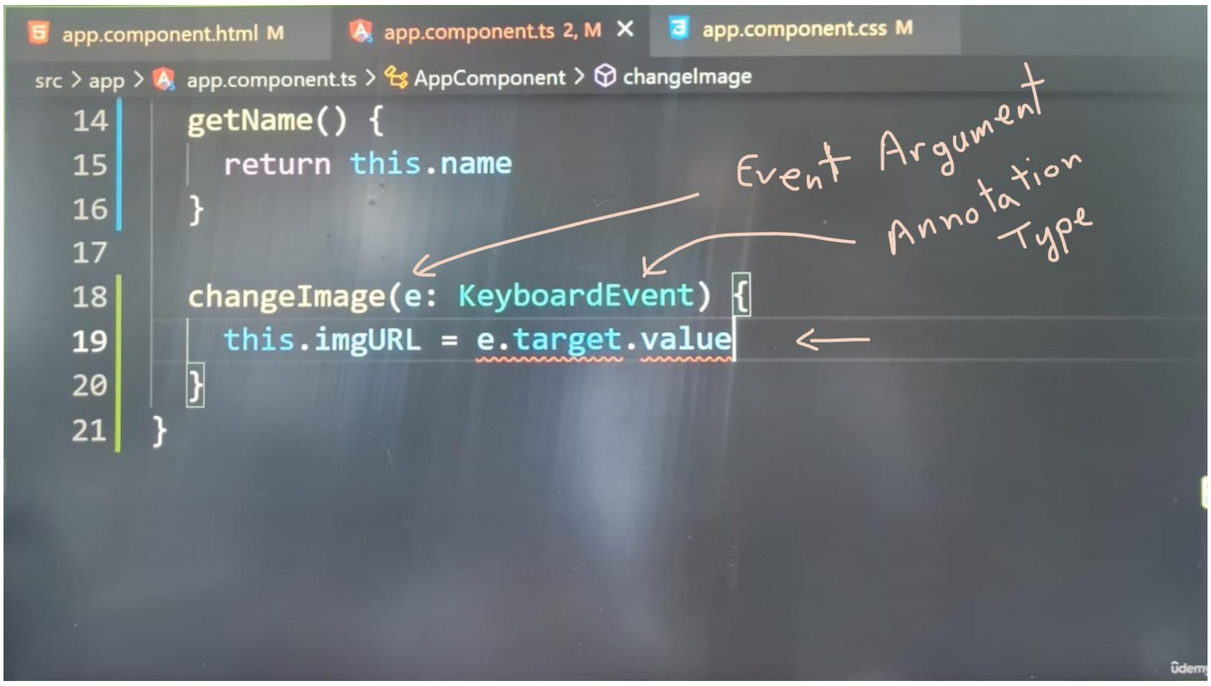
```
<input (keyup)="changeImage($event)" [value]="imgURL" />

<img [src]="imgURL" />

<p>Hello {{ name.toLocaleUpperCase() }}</p>
<p>Hello {{ getName() }}</p>
<p>{{ 15 + 13 }}</p>
```

Let's write the method for handling the **event** back in the component class, we're going to define the method:



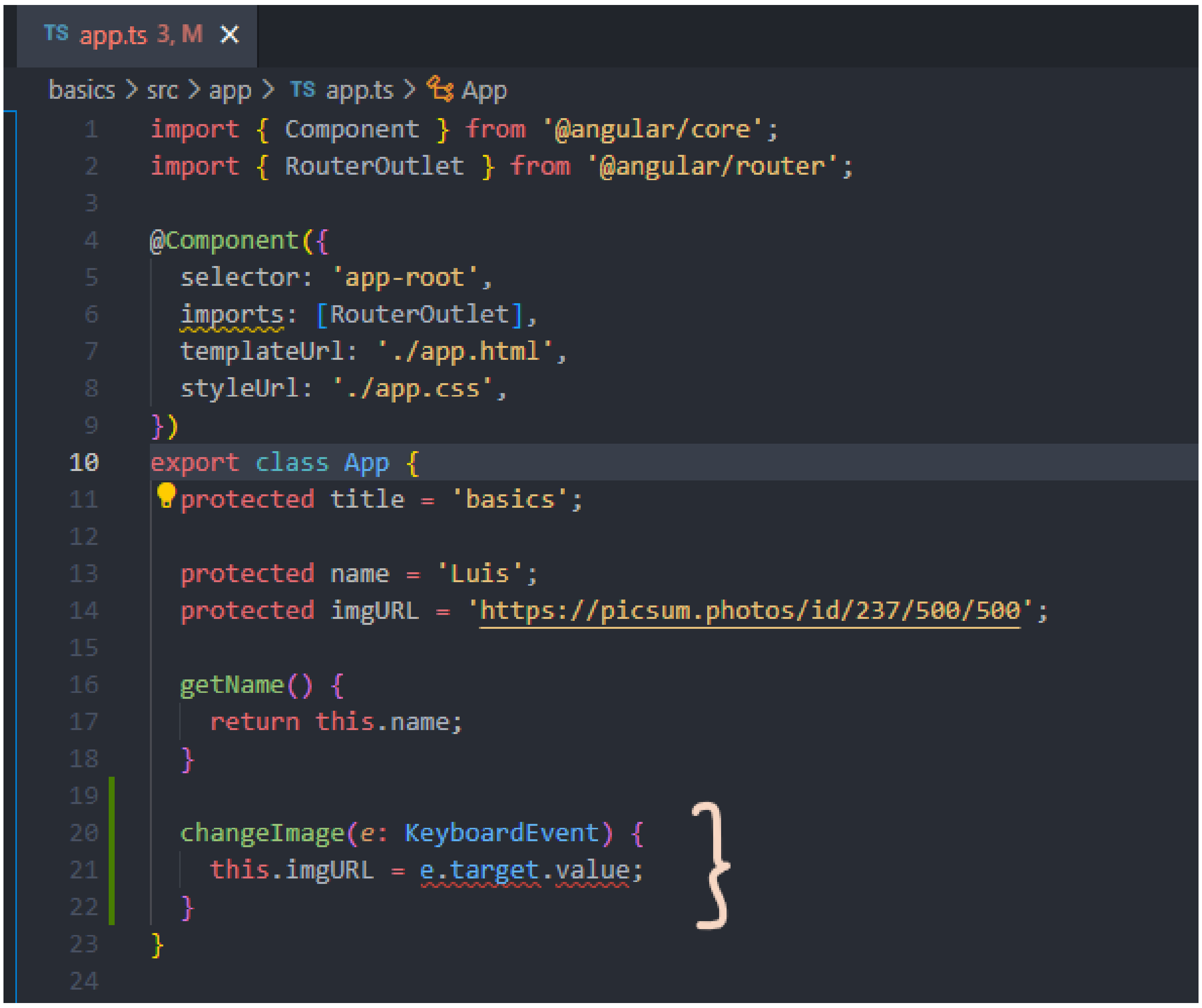
```
<input (keyup)="changeImage($event)" [value]="i

<img [src]="imgURL">

<p>Hello {{ name.toUpperCase() }}!</p>
<p>Hello {{ getName() }}!</p>
<p>{{ 15 + 13 }}</p>
```

We're going to accept the **event argument** as e. This argument should be annotated with the **KeyboardEvent** type:



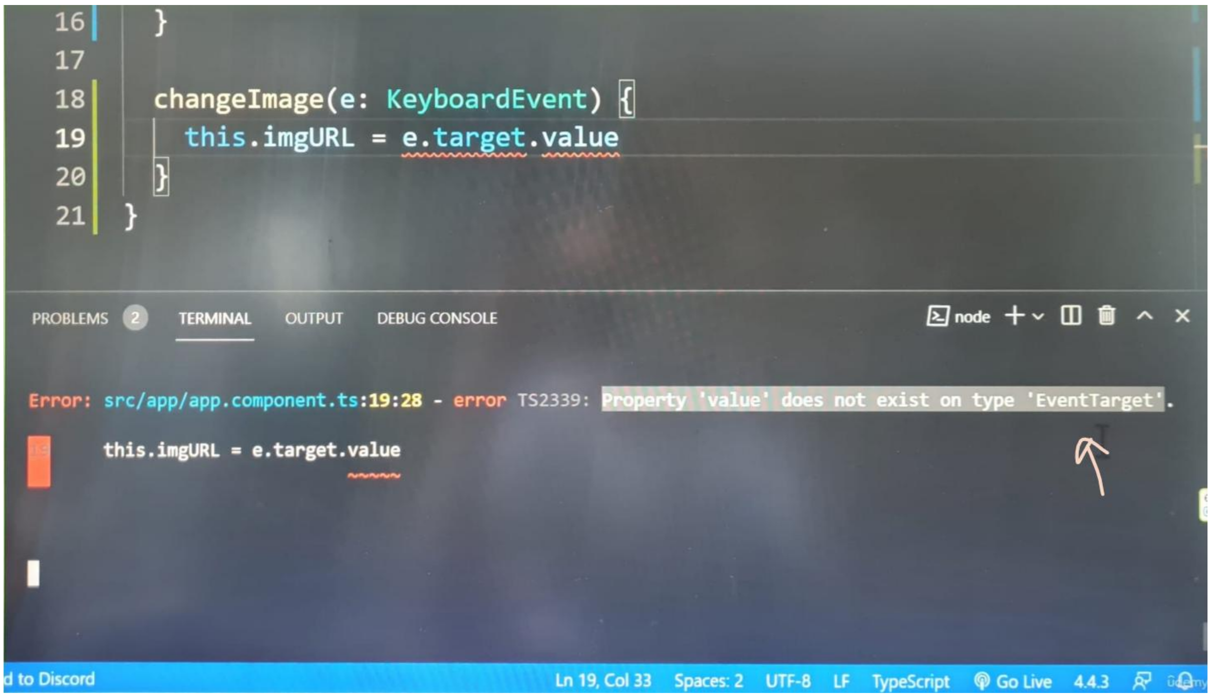
```
  getName() {
    return this.name
  }

  changeImage(e: KeyboardEvent) {
    this.imgURL = e.target.value
  }
}
```

On my PC with latest version of Angular 20.0.5 – app.ts:



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

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.html',
  styleUrl: './app.css',
})
export class App {
  protected title = 'basics';

  protected name = 'Luis';
  protected imgURL = 'https://picsum.photos/id/237/500/500';

  getName() {
    return this.name;
  }

  changeImage(e: KeyboardEvent) {
    this.imgURL = e.target.value;
  }
}
```

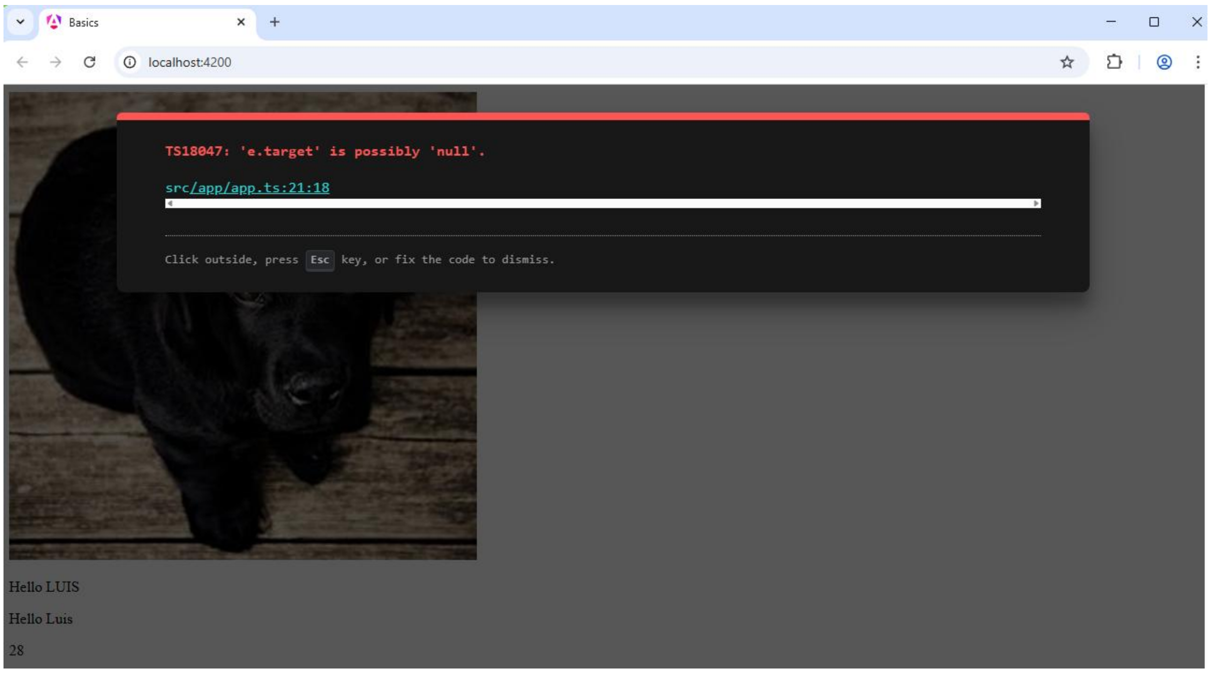
After saving the file, we're going to receive an error. In the console, the error is telling us the value property does not exist on the event target object, which doesn't make sense because we aren't using this object or so it seems:



```typescript
  }

  changeImage(e: KeyboardEvent) {
    this.imgURL = e.target.value
  }
}
```

```
Error: src/app/app.component.ts:19:28 - error TS2339: Property 'value' does not exist on type 'EventTarget'.

      this.imgURL = e.target.value
                             ~~~~~
```

On my PC with latest version of Angular 20.0.5 – Output:



```
TS18047: 'e.target' is possibly 'null'.

src/app/app.ts:21:18

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

```
Hello LUIS
Hello Luis
28
```

In the following lecture, let's explore this issue in depth.