ACCELERATING THROUGH ANGULAR 2

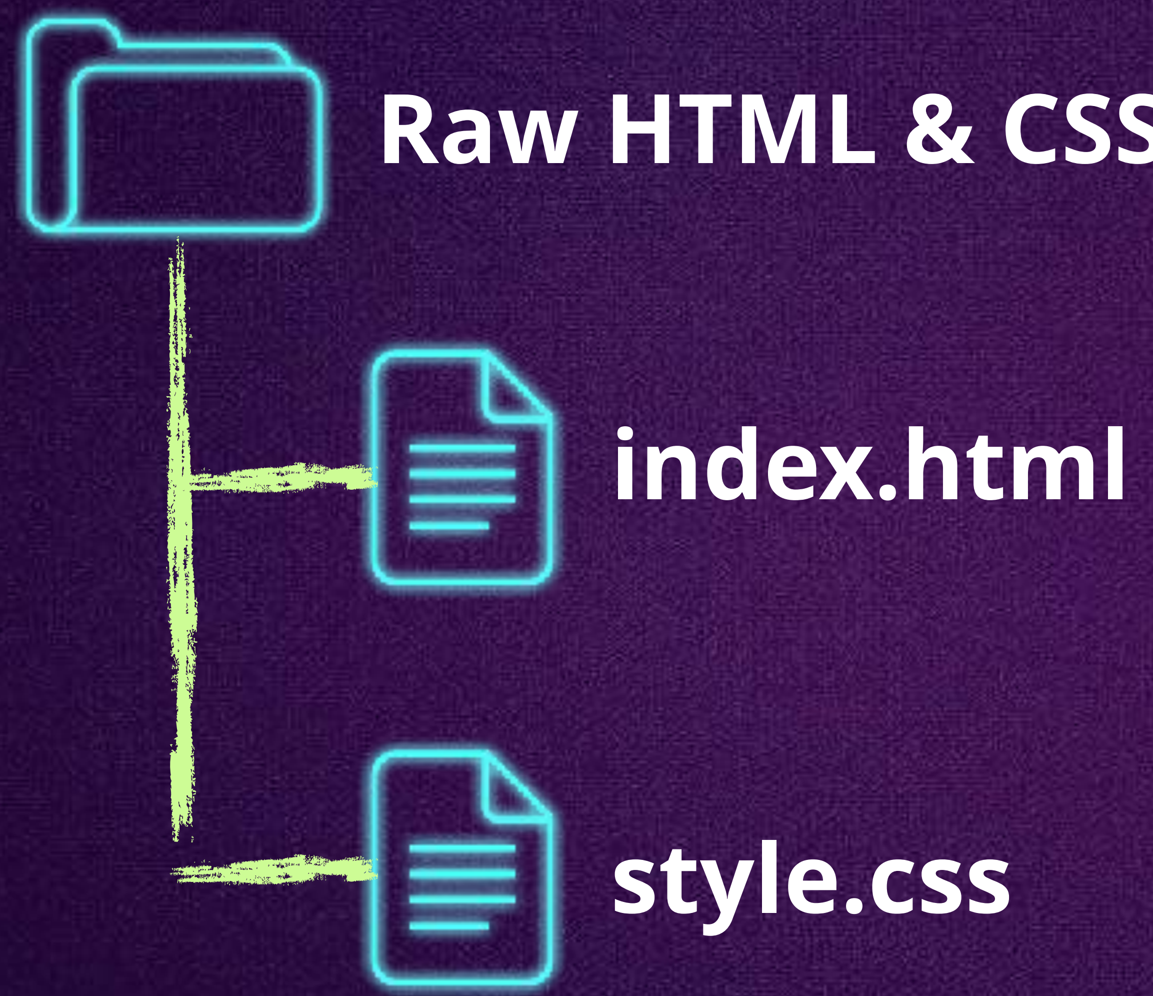
Level 4

Property & Class Binding

Section 1

Let’s Add Some Design

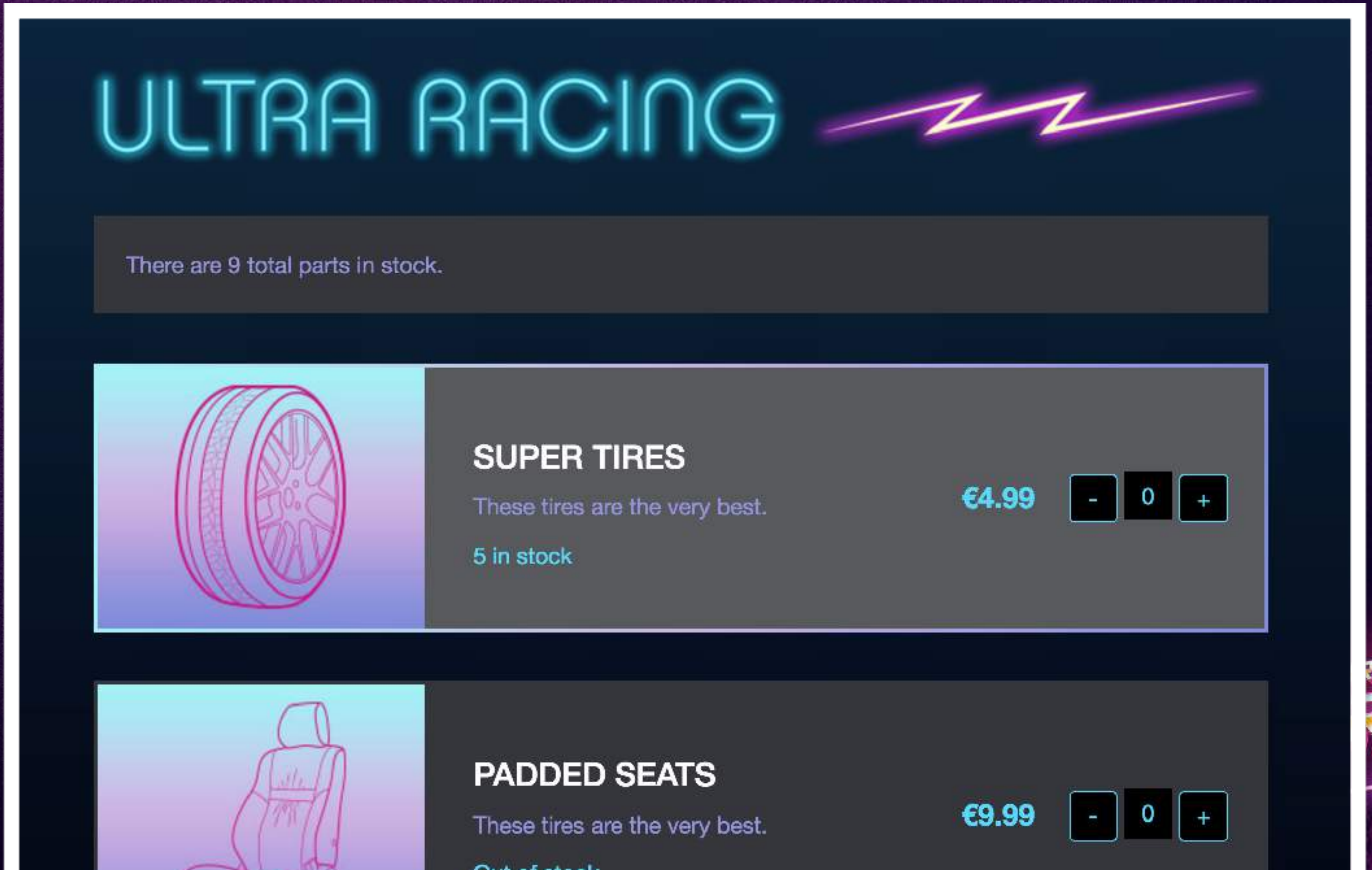
Not having to work with more complex HTML has been nice as we’ve learned Angular, but now we’re going to implement a better design.



Raw HTML & CSS From Designer

index.html

style.css



ULTRA RACING

There are 9 total parts in stock.

SUPER TIRES

These tires are the very best.

5 in stock

€4.99

PADDED SEATS

These tires are the very best.

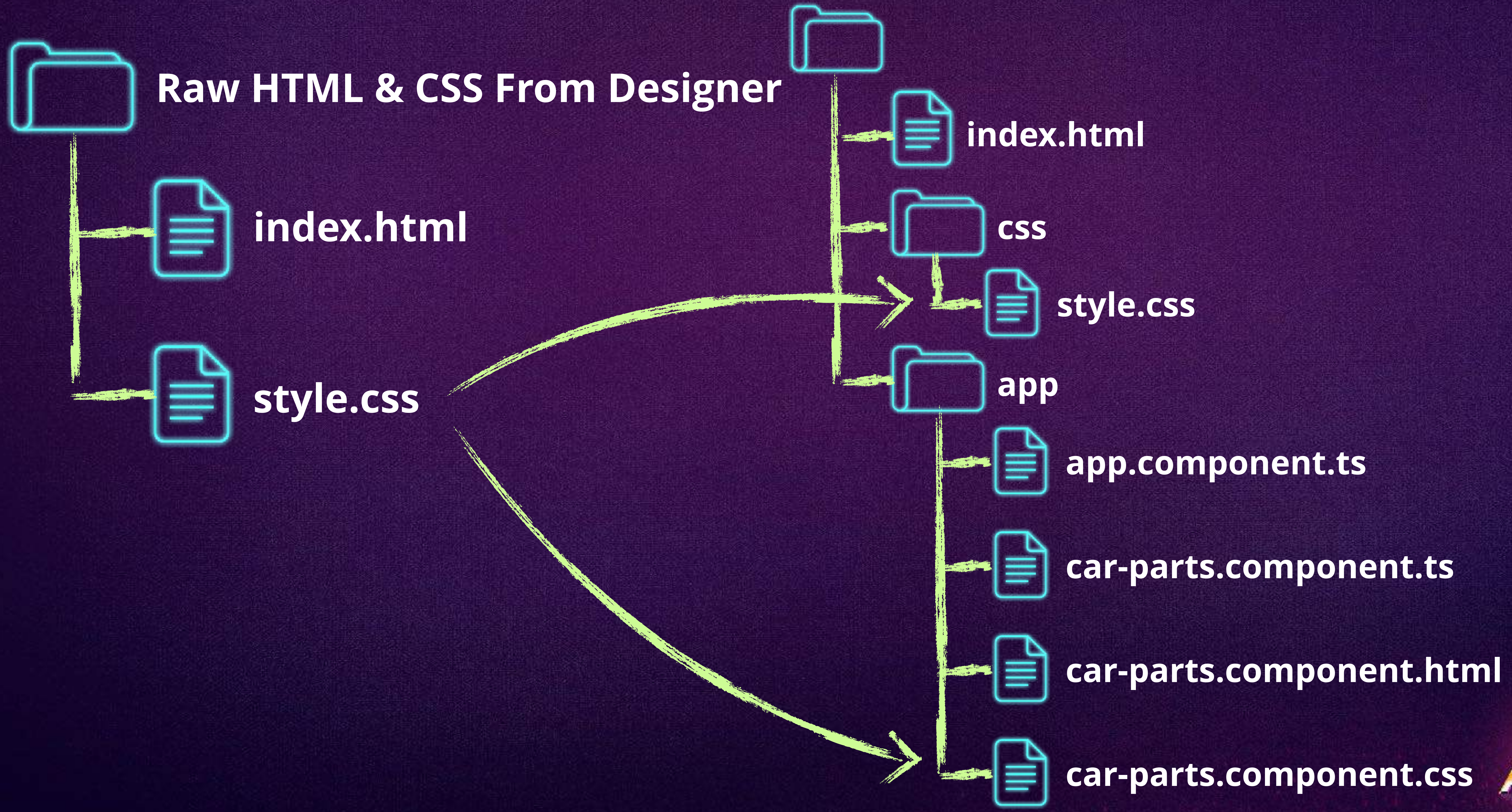
Out of stock

€9.99

We’ll be adding the images and quantity later this level.

Moving the CSS

Styles get put in a new CSS folder and the car-parts.component.css **for styles specific to that component.**



Raw HTML & CSS From Designer

index.html

style.css

index.html

css

style.css

app

app.component.ts

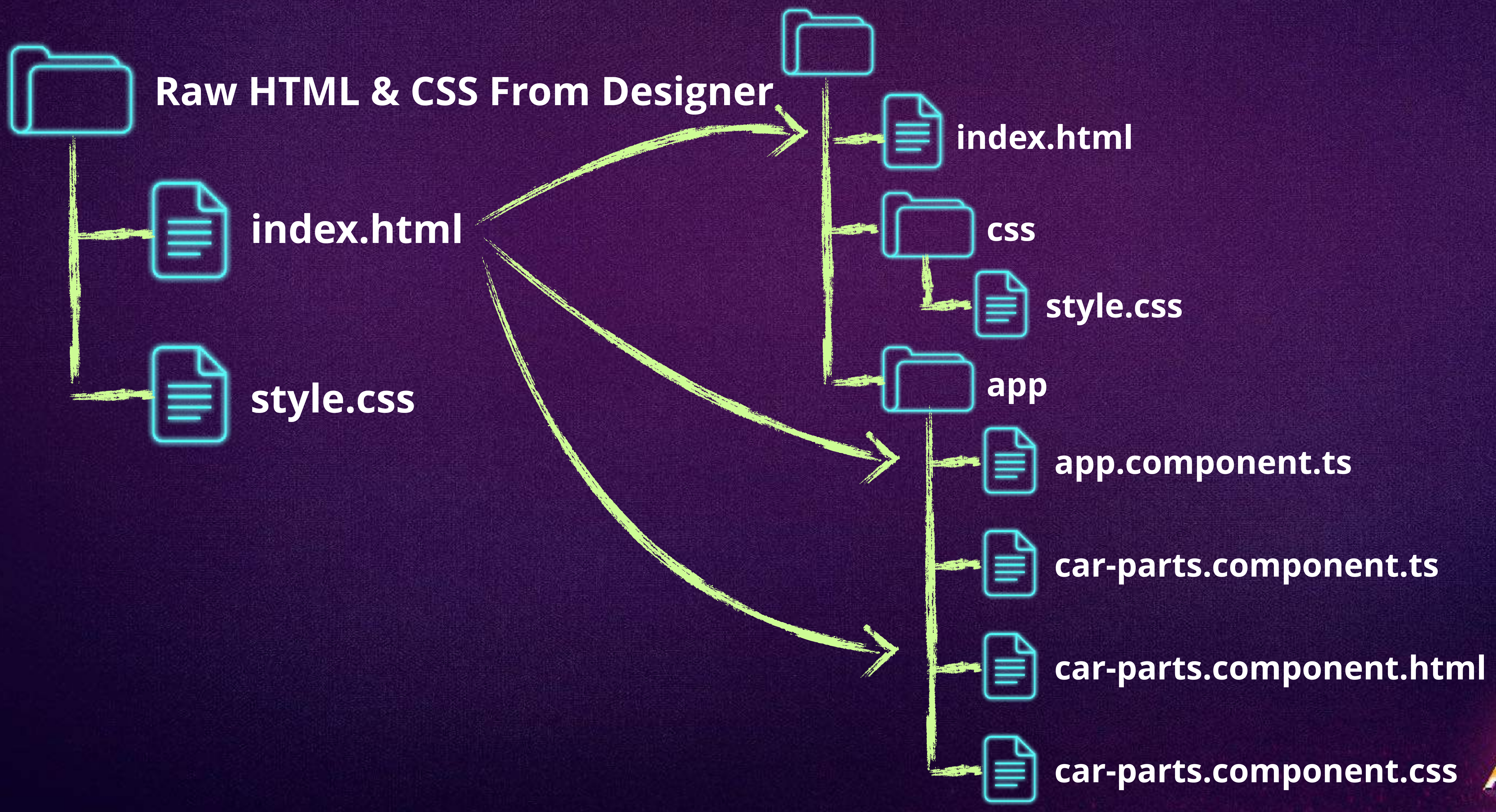
car-parts.component.ts

car-parts.component.html

car-parts.component.css

Splitting Up the HTML

HTML gets split up into three files.



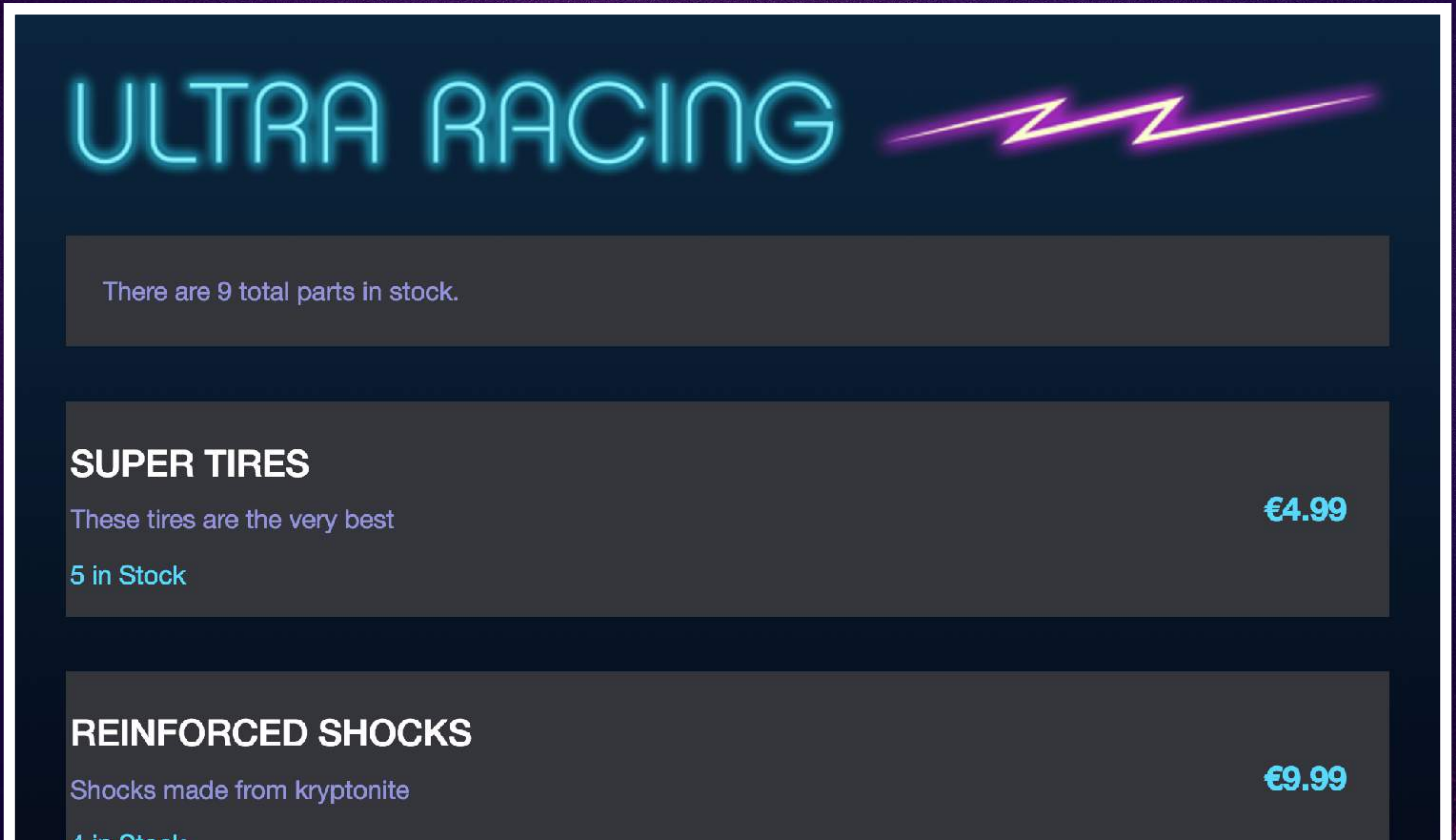
Raw HTML & CSS From Designer

- index.html
- style.css

- index.html
- css
  - style.css
- app
  - app.component.ts
  - car-parts.component.ts
  - car-parts.component.html
  - car-parts.component.css

Our Current App

Much better, but let's figure out how to bring images in.



ULTRA RACING

There are 9 total parts in stock.

SUPER TIRES

These tires are the very best

5 in Stock

€4.99

REINFORCED SHOCKS

Shocks made from kryptonite

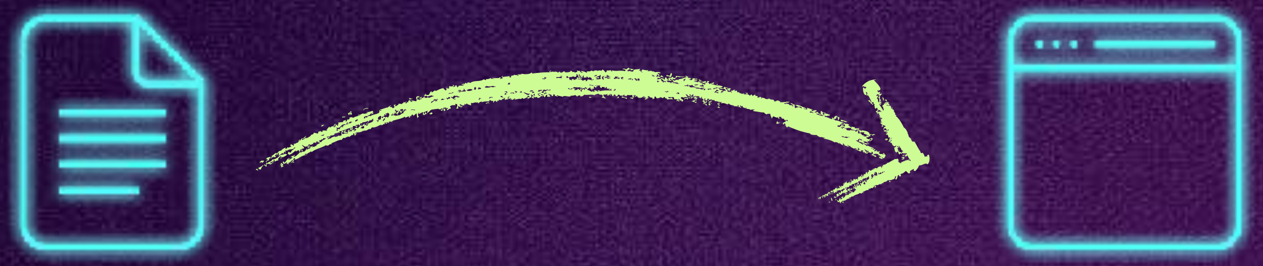
4 in Stock

€9.99

The Ways Data Can Flow

When using a web framework like Angular that abstracts your code from HTML, there are a few different ways that data can flow.

JavaScript to HTML

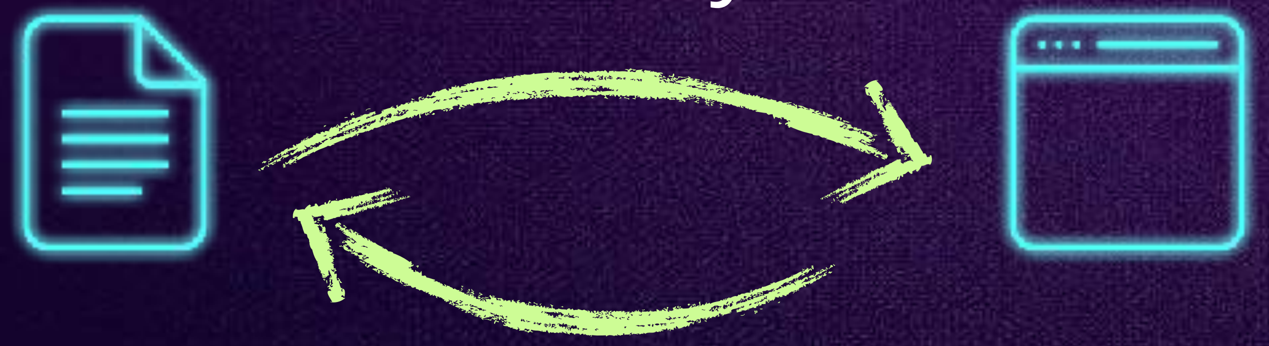


Like we've been doing with properties from our components

HTML to JavaScript

Like a mouse click, hover, or key press

Both Ways



Like a text box, that should stay in sync

Note: We're saying JavaScript here because our TypeScript turns into JavaScript.

JavaScript to HTML

In our application thus far, we’ve been sending all sorts of data from our components into our HTML using interpolation.

car-parts.component.html TypeScript

```
<li class="card" *ngFor="let carPart of carParts" >
  <div class="panel-body">
    <table class="product-info">
      <tr>
        <td>
          <h2>{{carPart.name | uppercase}}</h2>
           <p class="description">{{carPart.description}}</p>
           <p class="inventory" *ngIf="carPart.inStock > 0">{{carPart.inStock}} in Stock</p>
           <p class="inventory" *ngIf="carPart.inStock === 0">Out of Stock</p>
        </td>
        <td class="price">{{carPart.price | currency:'EUR':true }}</td>
      </tr>
    </table>
  </div>
</li>
```

When code is interpolated, the properties are read from the component and strings are printed.

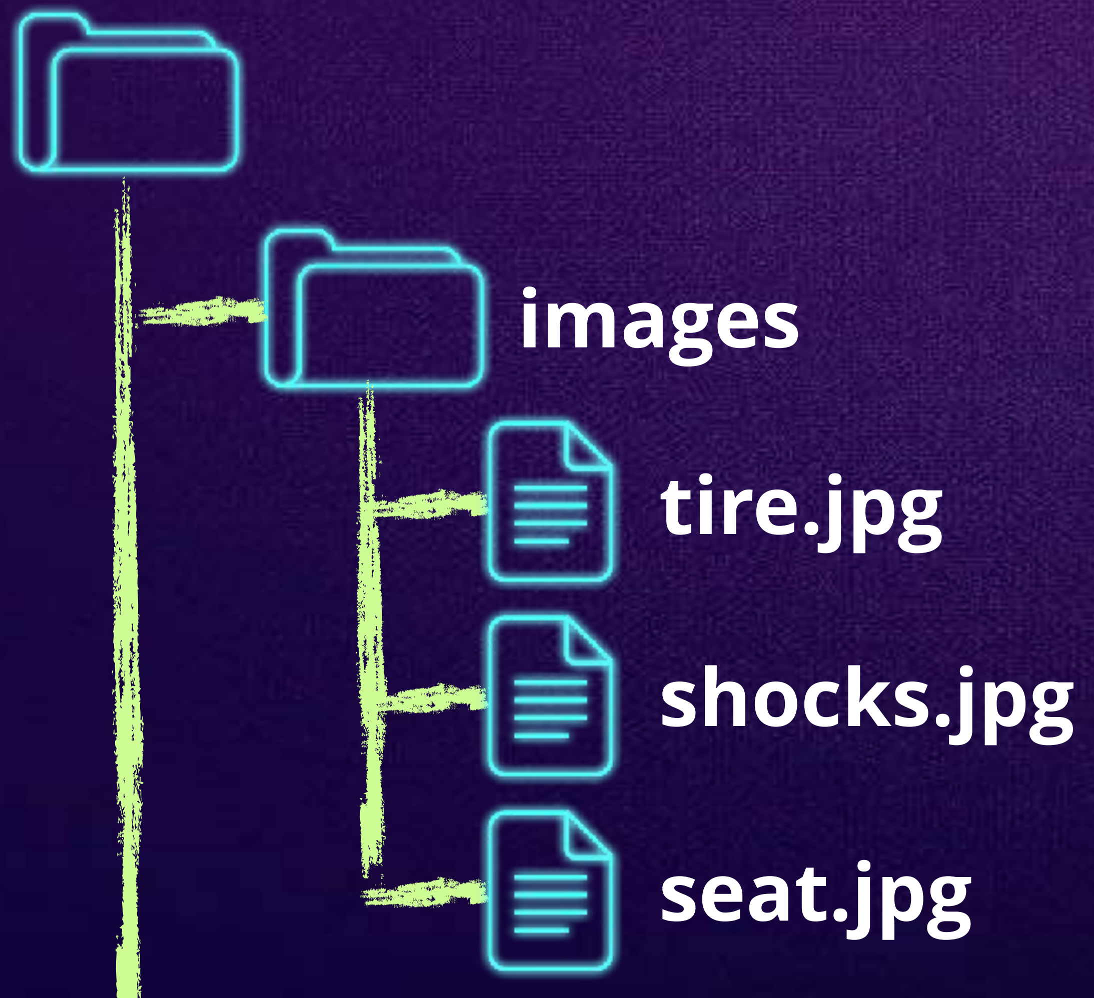
So, how would we add an image tag with a dynamic image?

Adding Images to Our Model

We will add this property to our model, add new files, and add them to our mock data.

car-part.ts TypeScript

```
export class CarPart {
  id: number;
  name: string;
  description: string;
  inStock: number;
  price: number;
  image: string;
}
```



images

tire.jpg

shocks.jpg

seat.jpg

mocks.ts TypeScript

```
import { CarPart } from './car-part';

export let CARPARTS: CarPart[] = [{
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5,
    "price": 4.99,
    "image": "/images/tire.jpg"
  },
  {
    "id": 2,
    "name": "Reinforced Shocks",
    "description": "Shocks made from kryptonite",
    "inStock": 4,
    "price": 9.99,
    "image": "/images/shocks.jpg"
  }, { ... } ];
```

Adding an Image to Our Template

We could try adding our image onto our page using interpolation.

car-parts.component.html TypeScript

```
<li class="card" *ngFor="let carPart of carParts" >
  <div class="panel-body">
    <div class="photo">
      <img src="{{carPart.image}}">
    </div>
    <table class="product-info">
      <tr>
        <td>
          <h2>{{carPart.name | uppercase}}</h2>
          ...
```

This would work just fine.

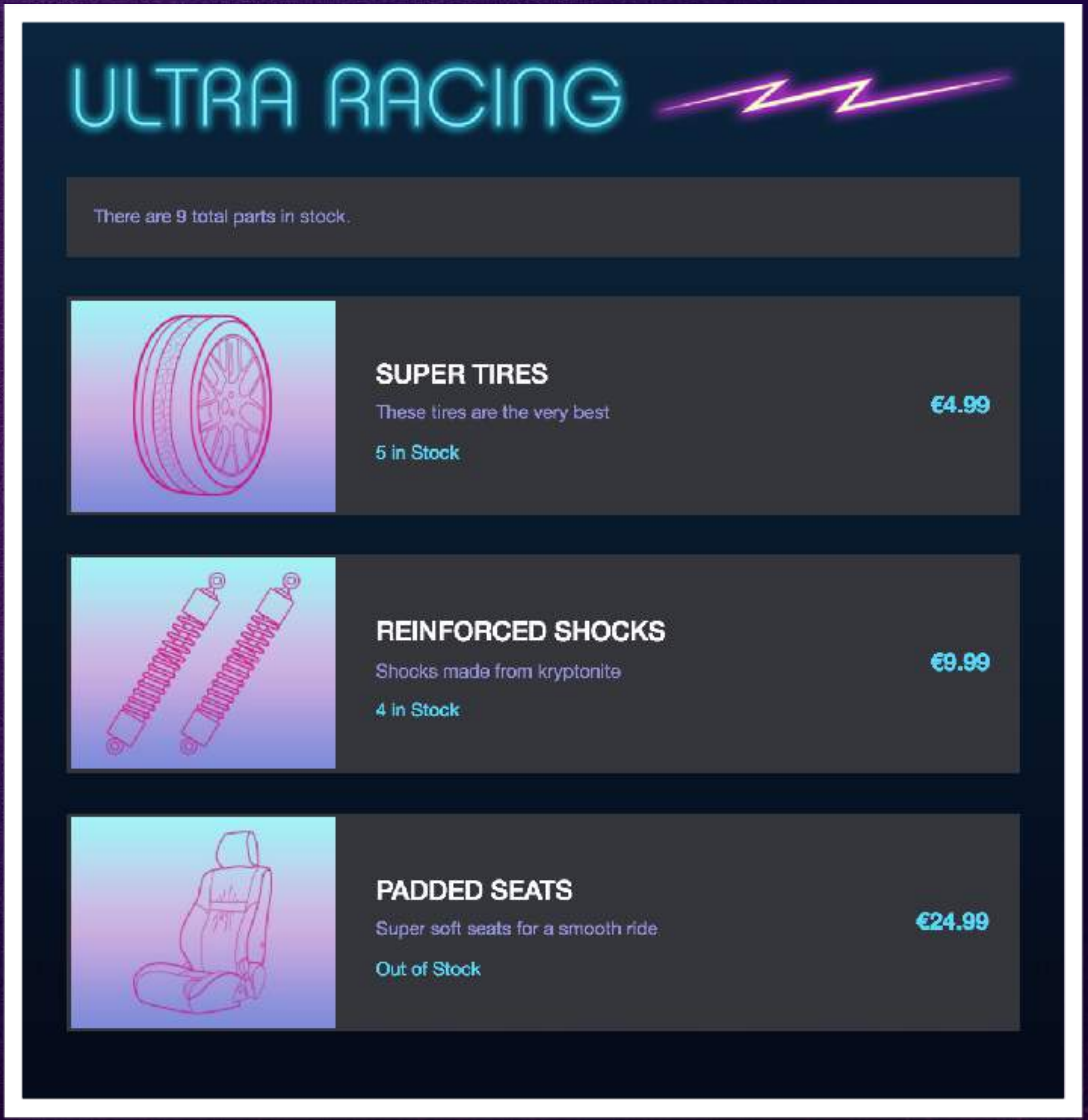
However, there’s an alternative syntax we can use when we want to set DOM element property values.

Introducing Property Binding

Property binding allows us to glue component properties to DOM element properties.

car-parts.component.html TypeScript

```
<li class="card" *ngFor="let carPart of carParts" >
  <div class="panel-body">
    <div class="photo">
      <img [src]="carPart.image">
    </div>
    <table class="product-info">
      <tr>
        <td>
          <h2>{{carPart.name | uppercase}}</h2>
          ...
```



There are 9 total parts in stock.

SUPER TIRES
These tires are the very best
5 in Stock
€4.99

REINFORCED SHOCKS
Shocks made from kryptonite
4 in Stock
€9.99

PADDED SEATS
Super soft seats for a smooth ride
Out of Stock
€24.99

Notice the square brackets and no curly braces!

The square brackets tell Angular to set this DOM element property to our component property.

And if the component property changes, update this.

Additional Property Binding Examples

We can bind to any DOM element property. How do you think we could bind to these?

```
<div hidden>secret</div>
```

```
<button disabled>Purchase</button>
```

```
<img alt="Image Description">
```

Our previous solution

```
<img [src]="carPart.image">
```

Additional Property Binding Examples

All we need to do is add brackets and specify a component property.

```
<div [hidden]="!user.isAdmin">secret</div>
```

```
<button [disabled]="isDisabled">Purchase</button>
```

```
<img [alt]="image.description">
```

Our previous solution

```
<img [src]="carPart.image">
```

Adding a Featured Car Part

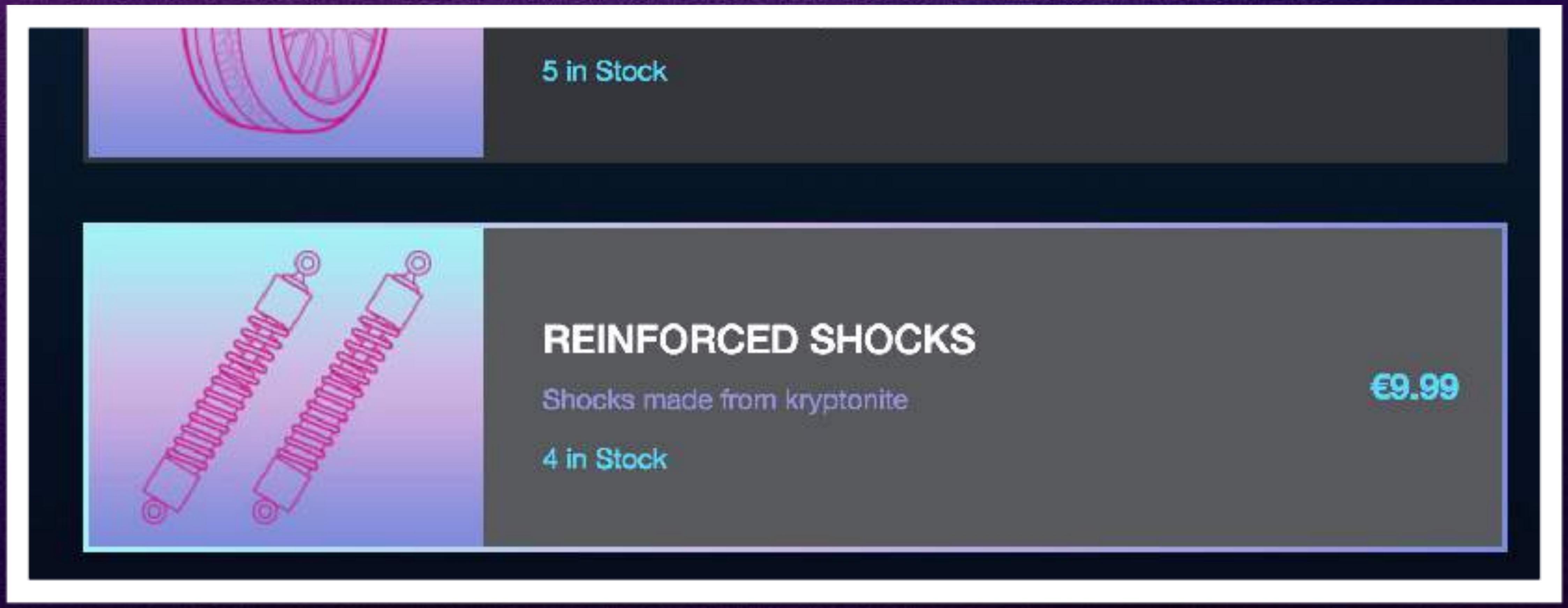
If a car part is marked as “featured,” we want to add a class of featured **to it.**

car-parts.component.css CSS

```css
...
.featured {
  background: #57595D;
  -webkit-border-image: -webkit-linear-gradient(right, #818fd8 0%, #cbb4e2 50%, #a6f2f5 100%);
       -o-border-image: linear-gradient(to left, #818fd8 0%, #cbb4e2 50%, #a6f2f5 100%);
          border-image: linear-gradient(to left, #818fd8 0%, #cbb4e2 50%, #a6f2f5 100%);
  border-image-slice: 1;
}
```

Here is the `featured` class, which adds a lighter background and a gradient border.

How do we add functionality to sometimes add this `featured` class?



5 in Stock

REINFORCED SHOCKS

Shocks made from kryptonite

4 in Stock

€9.99

Adding a Featured Property & Data

We need to add a new property to our car-part.ts **model and add mock data for it.**

car-part.ts TypeScript

```
export class CarPart {
  ...
  image: string;
  featured: boolean;
}
```

mocks.ts TypeScript

```
export let CARPARTS: CarPart[] = [{
    "id": 1,
    ...
    "featured": false
  },
  {
    "id": 2,
    ...
    "featured": true
  },
  {
    "id": 3,
    ...
    "featured": false
}];
```

Next, we need to conditionally add a class if this property is true.

Using a Class Property Binding

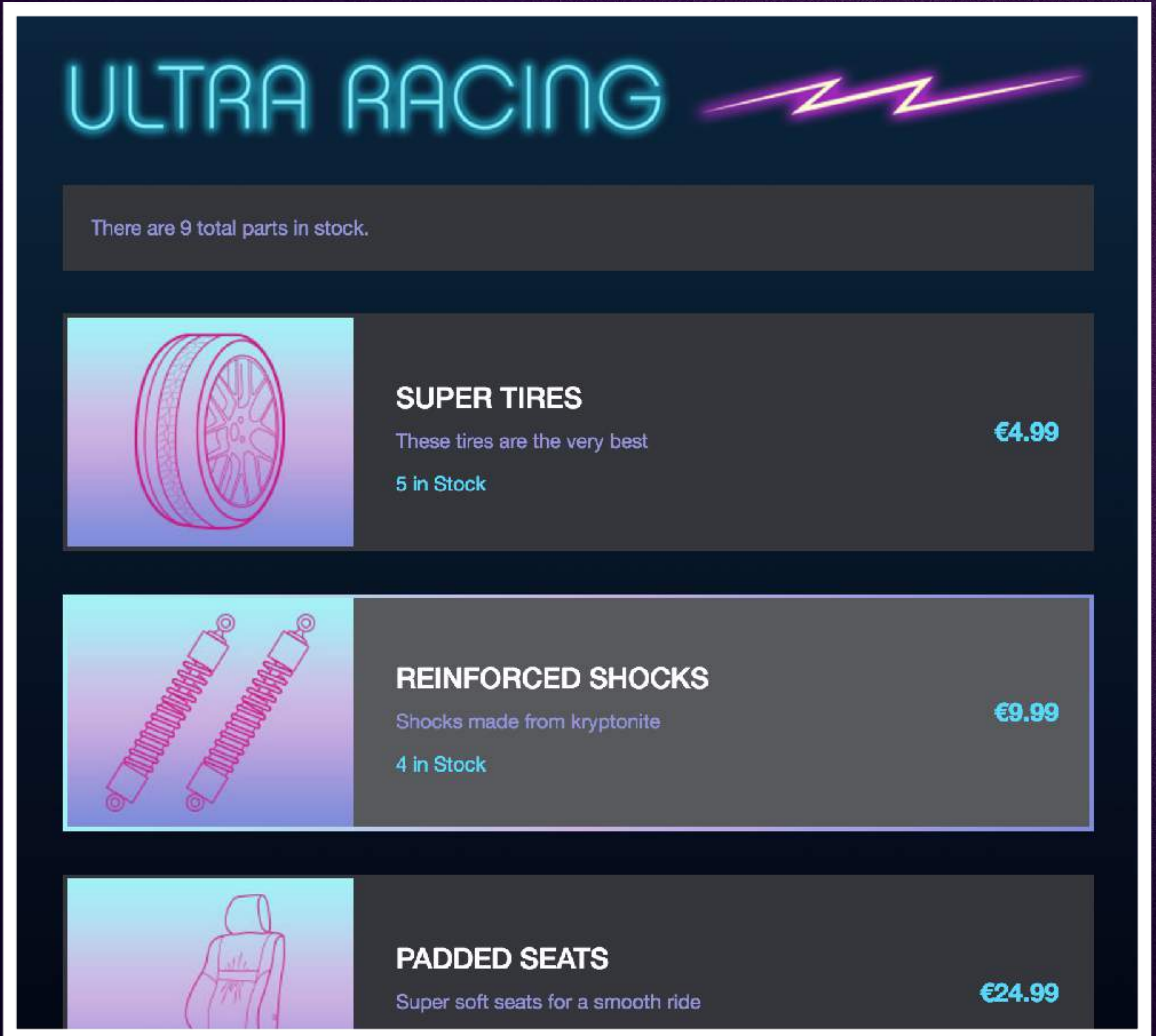
There’s a unique syntax for binding to a class.

car-parts.component.html TypeScript

```
<ul>
  <li class="card" *ngFor="let carPart of carParts" [class.featured]="carPart.featured">
    <div class="panel-body">
      ...
    </div>
  </li>
</ul>
```

If `carPart.featured` is true, then the `featured` class is added. If `carPart.featured` is false, then the `featured` class is removed.

Looking Into the Web Page



ULTRA RACING

There are 9 total parts in stock.

SUPER TIRES
These tires are the very best
5 in Stock
€4.99

REINFORCED SHOCKS
Shocks made from kryptonite
4 in Stock
€9.99

PADDED SEATS
Super soft seats for a smooth ride
€24.99

```
[class.featured]="carPart.featured"
```

Looking at the source, we see that the element and the class are properly scoped.

```
<li _ngcontent-opf-2 class="card featured">
```

```
.featured[_ngcontent-opf-2] {
    background: #57595D;
    ...
}
```

How Not to Use Class Binding

You might be tempted to bind directly to the class element property:

```
<div [class]="property">
```

✗ This will overwrite all classes.

```
<div [class.name]="property">
```

✓ This will only add/remove the specific class.

Class names with dashes also work fine.

```
<div [class.my-name]="property">
```

What’d We Learn?

- Property binding allows us to bind component properties to any DOM element properties.
- Any update to the component property value will update the DOM property, but not vice versa — that’s why it’s “one-way binding.”
- Class binding allows us to specify a CSS class to add to a DOM element if a component property is true.

ACCELERATING THROUGH
ANGULAR 2

Level 4

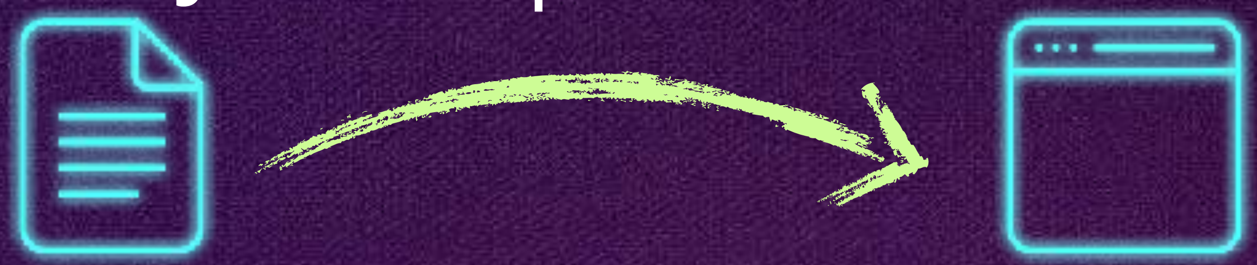
Event Binding

Section 2

Types of Data Binding

Property Binding
Class Binding

JavaScript to HTML

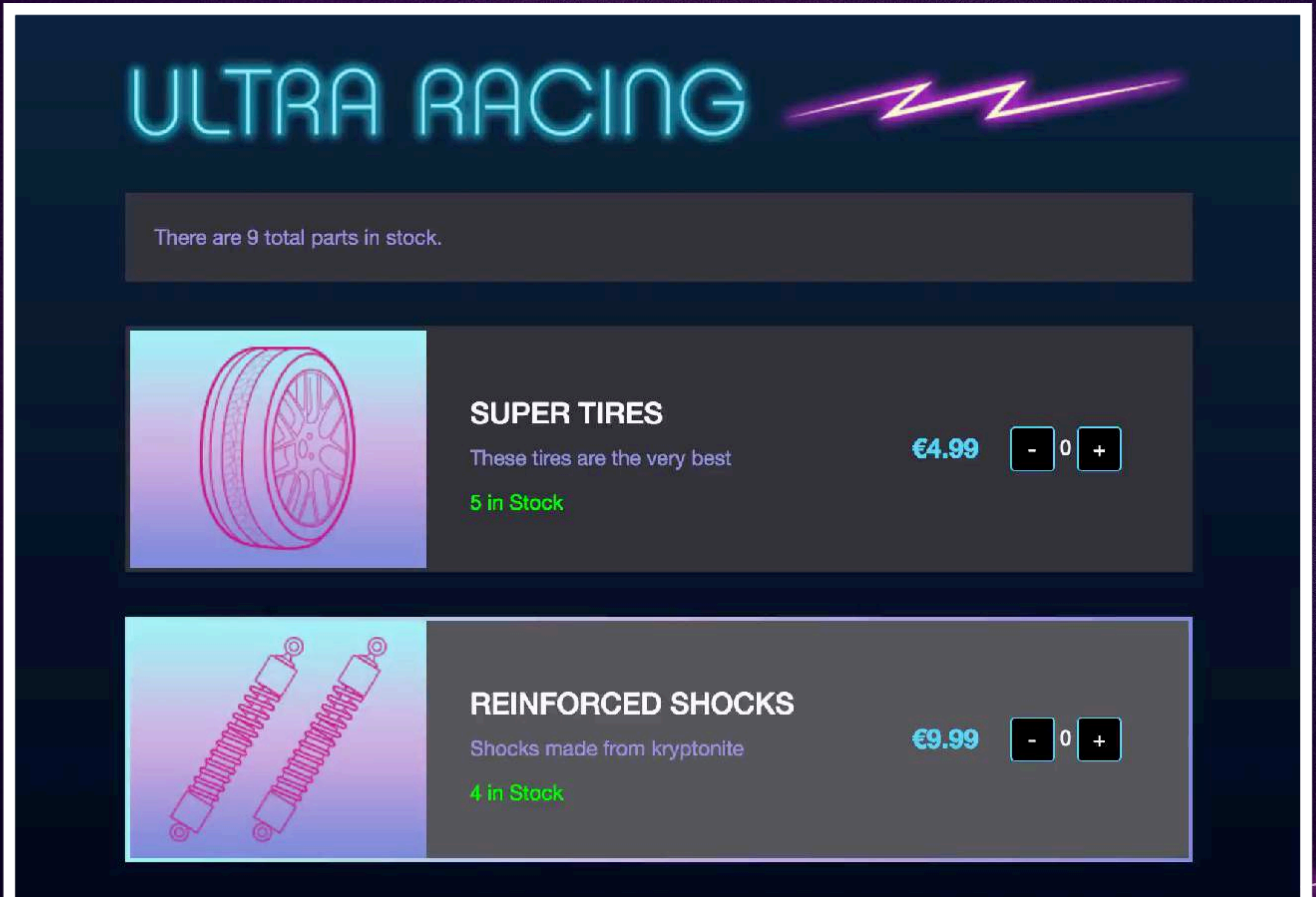


Event Binding

HTML to JavaScript

Like a mouse click, hover, or key press

Adding Events to Our Page



ULTRA RACING

There are 9 total parts in stock.

SUPER TIRES

These tires are the very best

5 in Stock

€4.99

REINFORCED SHOCKS

Shocks made from kryptonite

4 in Stock

€9.99

Adding a Quantity Property & Data

We need to add a new property to our car-part.ts **model and add mock data for it.**

car-part.ts TypeScript

```
export class CarPart {
  ...
  image: string;
  featured: boolean;
  quantity: number;
}
```

mocks.ts TypeScript

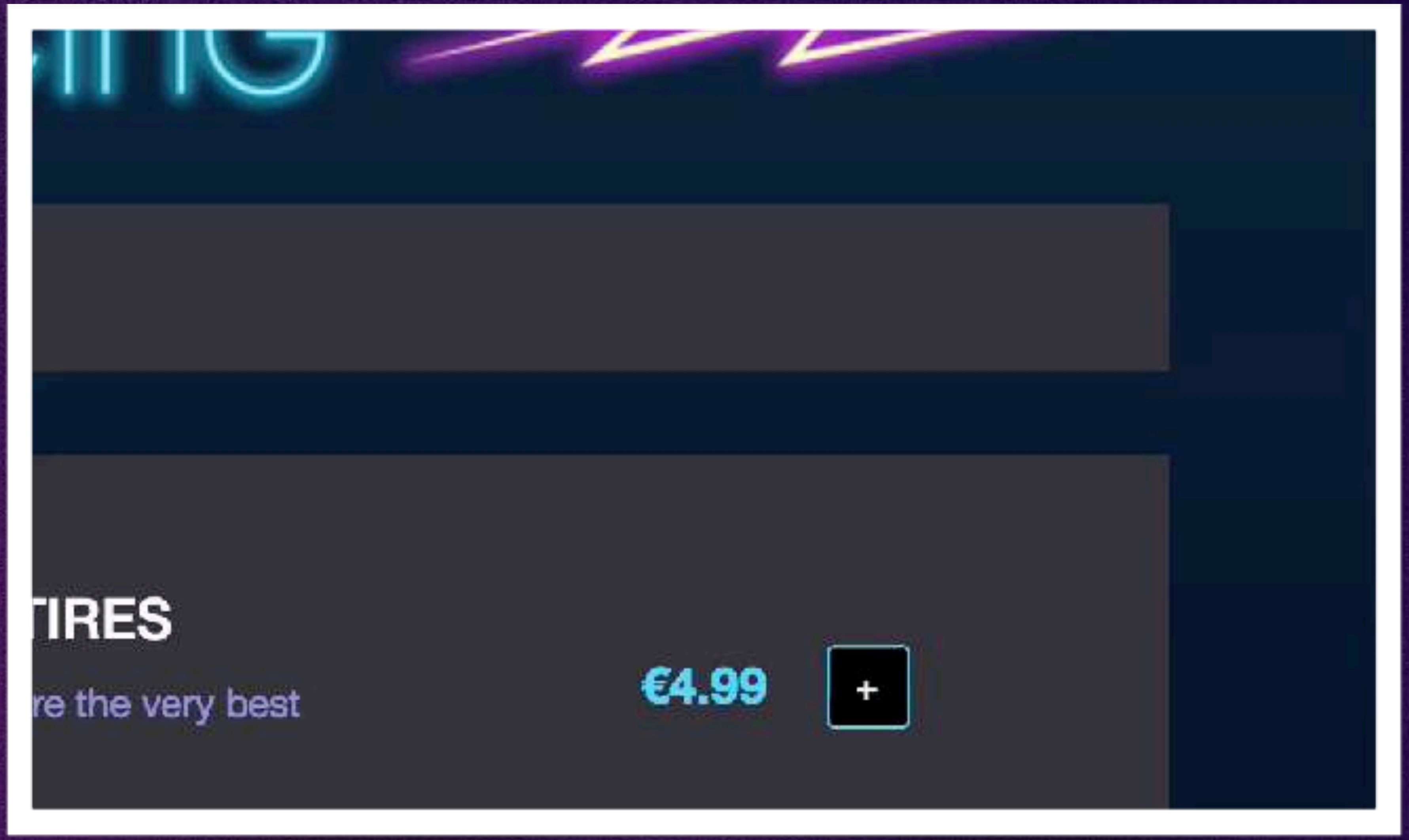
```
export let CARPARTS: CarPart[] = [{
    "id": 1,
    ...
    "featured": false,
    "quantity": 0
  }, { ... }, { ... }];
```

Adding a Simple Button

car-parts.component.ts TypeScript

```typescript
...
export class CarPartsComponent {
  ...

  upQuantity() {
    alert("You Called upQuantity");
  }
  ...
```



To capture an event from our template, we wrap the name of the event we want to listen to in parentheses and specify the method to call.

car-parts.component.html HTML

```html
 ...
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
     <button class="increase" (click)="upQuantity()">+</button>
 ...
```

Making It Actually Work

Now let's use the carPart.quantity **that we have on each car part.**

car-parts.component.html HTML

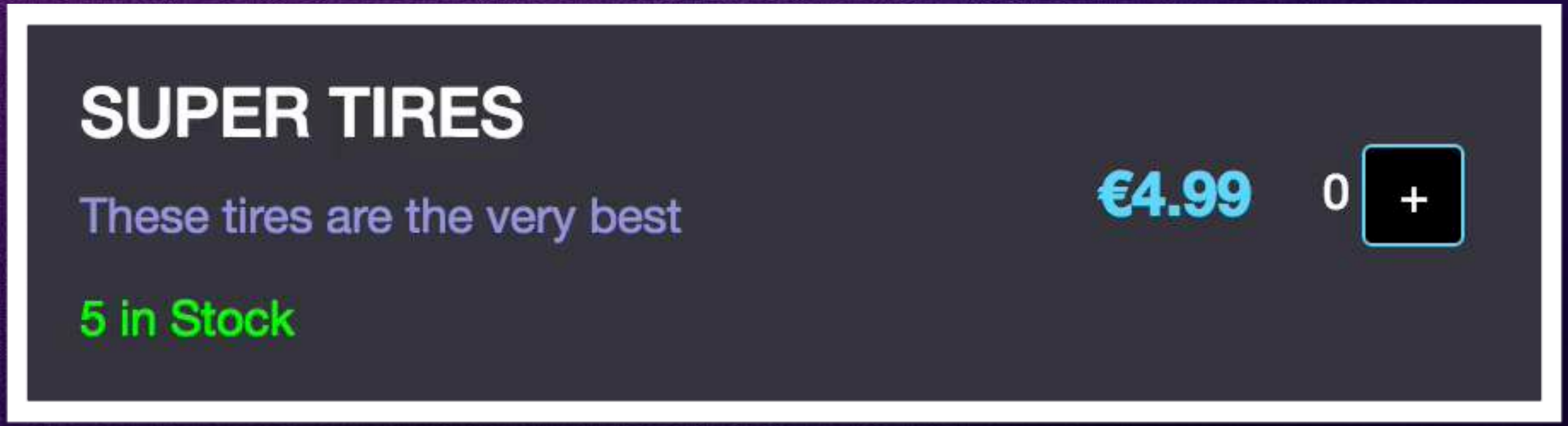
```html
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
      {{carPart.quantity}}
      <button class="increase" (click)="upQuantity(carPart)">+</button>
```

We need to send in the current carPart.

car-parts.component.ts TypeScript

```typescript
export class CarPartsComponent {
  ...

  upQuantity(carPart) {
    carPart.quantity++;
  }
```



SUPER TIRES

These tires are the very best

5 in Stock

€4.99

0

Uh-oh — we can increase the quantity we want beyond what we have in stock.

Limited Incrementing

We shouldn’t be able to add more quantity than we have in stock.

car-parts.component.html **HTML**

```html
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
     {{carPart.quantity}}
     <button class="increase" (click)="upQuantity(carPart)">+</button>
```

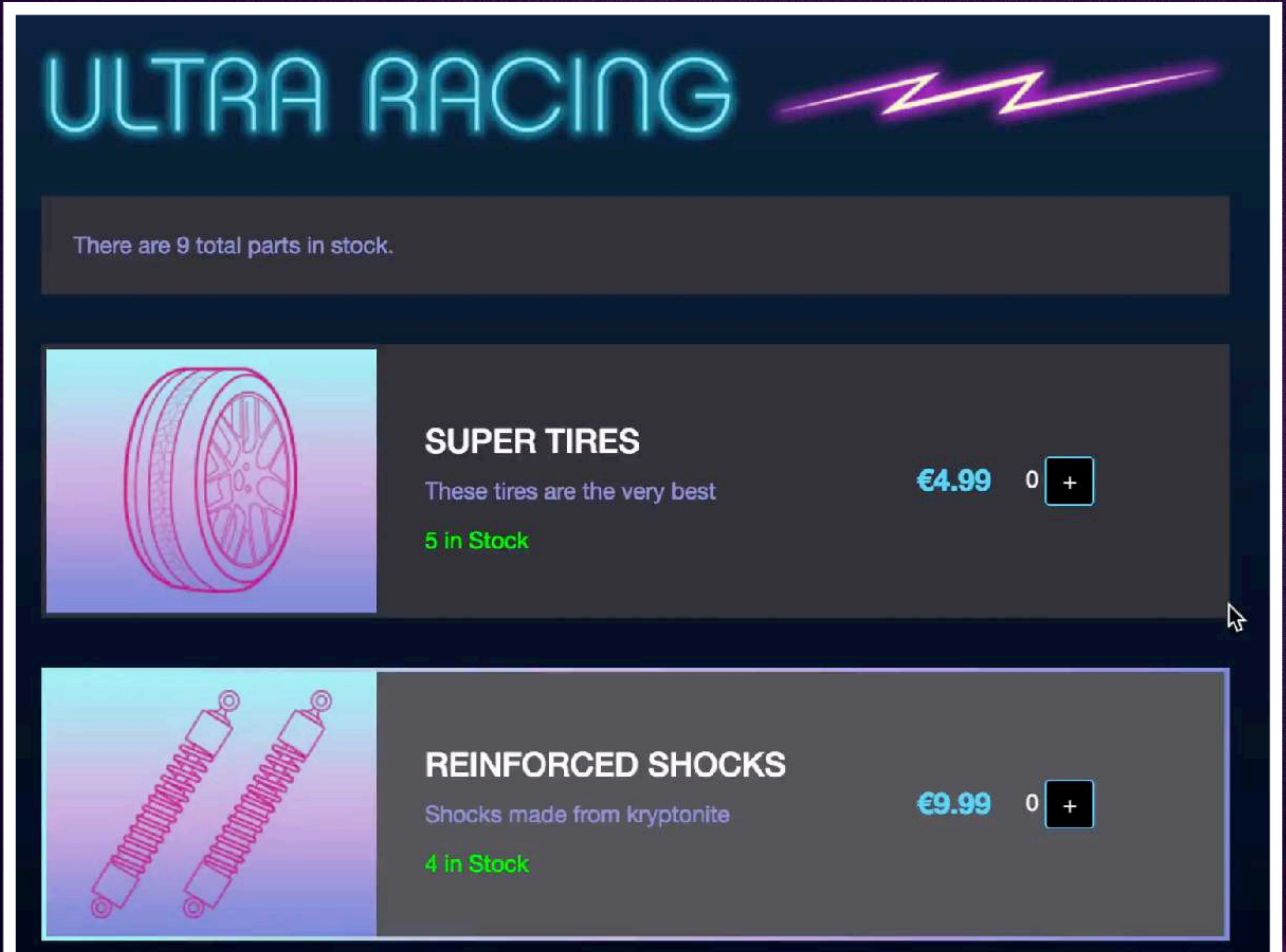
car-parts.component.ts **TypeScript**

```typescript
export class CarPartsComponent {
  ...

  upQuantity(carPart) {
    if (carPart.quantity < carPart.inStock) carPart.quantity++;
  }
```

Only add quantity if current quantity is less than we have in stock.

Now With Proper Limits



ULTRA RACING

There are 9 total parts in stock.

SUPER TIRES

These tires are the very best

5 in Stock

REINFORCED SHOCKS

Shocks made from kryptonite

4 in Stock

Adding Our Decrease Button

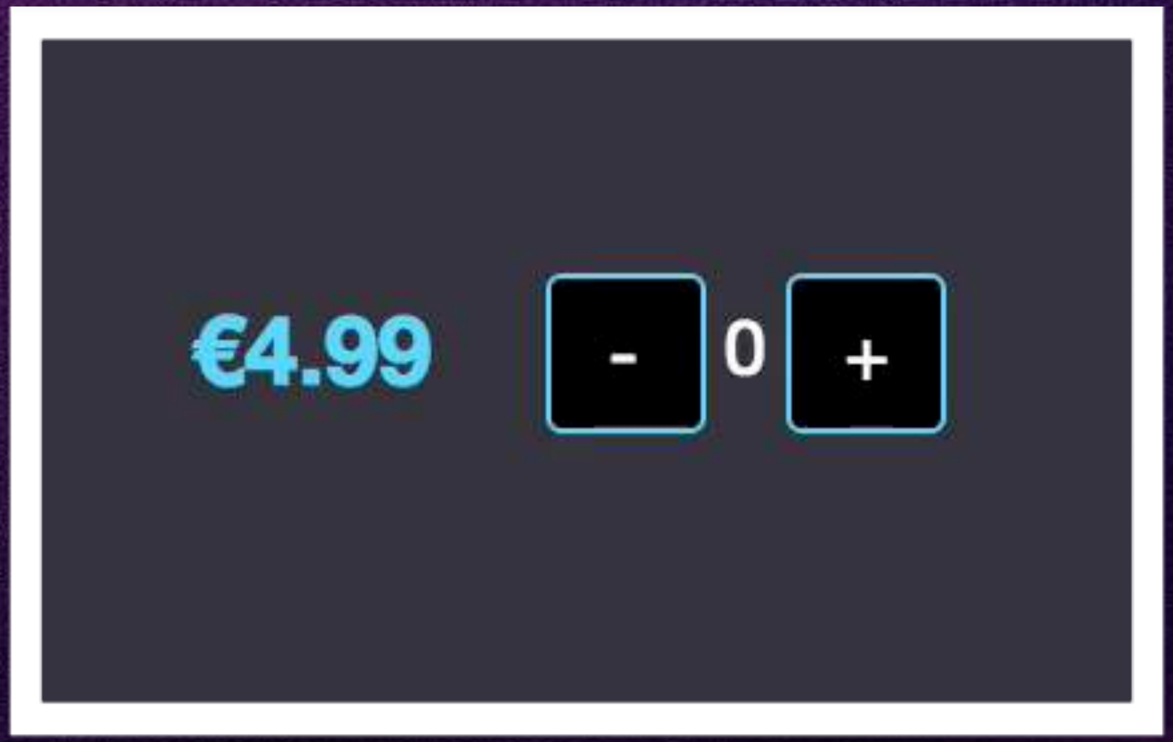
We should be able to decrease the quantity, but not below zero.

car-parts.component.html HTML

```html
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
     <button class="decrease" (click)="downQuantity(carPart)">-</button>
     {{carPart.quantity}}
     <button class="increase" (click)="upQuantity(carPart)">+</button>
```

car-parts.component.ts TypeScript

```typescript
export class CarPartsComponent {
  ...
  downQuantity(carPart) {
    if (carPart.quantity != 0) carPart.quantity--;
  }
}
```



Only subtract quantity if current quantity is not zero.

Other Events to Listen For

Any standard DOM event can be listened for by wrapping it in parentheses and removing the “on” at the beginning of the word.

```
<div (mouseover)="call()">
```

```
<input (blur)="call()">
```

```
<input (focus)="call()">
```

```
<input type="text" (keydown)="call()">
```

```
<form (submit)="call()">
```

Getting Additional Event Data

Sometimes you need additional event data, like which key is pressed or where the mouse is on the screen. This is what the Angular event object is for.

```
<input type="text" (keydown)="showKey($event)">
```

```
showKey(event) {
  alert(event.keyCode);
}
```

We can send the $event *object into our methods.*

```
<h2 (mouseover)="getCoord($event)">Hover Me</h2>
```

```
getCoord(event) {
  console.log(event.clientX + ", " + event.clientY);
}
```

We could also call `event.preventDefault();` to prevent a clicked link from being followed or a form from being submitted.

What’d We Learn?

- Event binding allows us to listen to any DOM event and call a component method when it’s triggered.
- To listen to any event, we need to remove the “on” in front of the word, wrap it in parentheses, and specify a component method to call.
- If we need to access the event object, we can pass it in to our component method with `$event`.

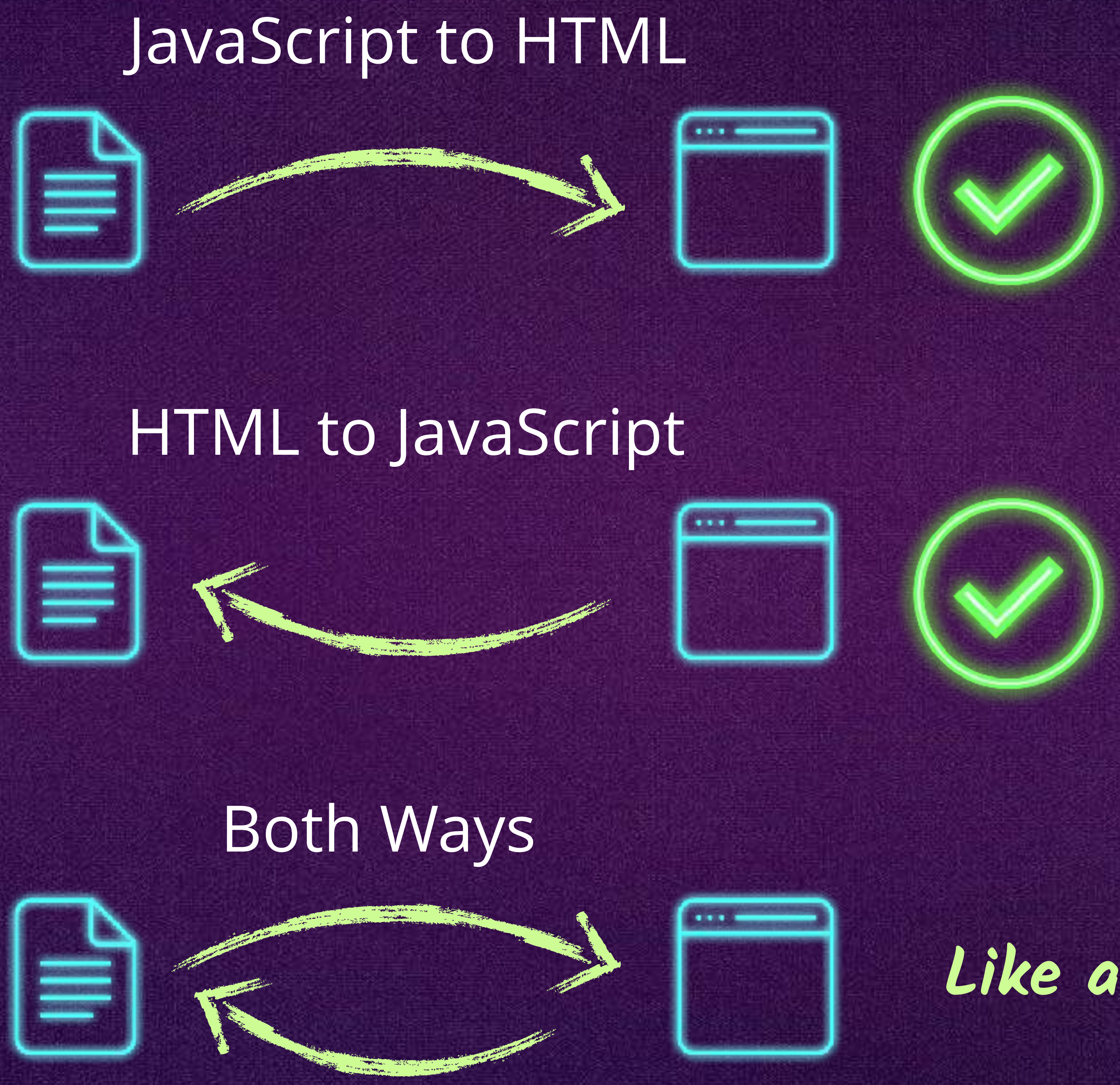
ACCELERATING THROUGH
ANGULAR 2

Level 4

Two-way Binding

Section 3

Types of Data Binding



JavaScript to HTML

Property Binding
Class Binding

HTML to JavaScript

Event Binding

Both Ways

Like a text box, that should stay in sync

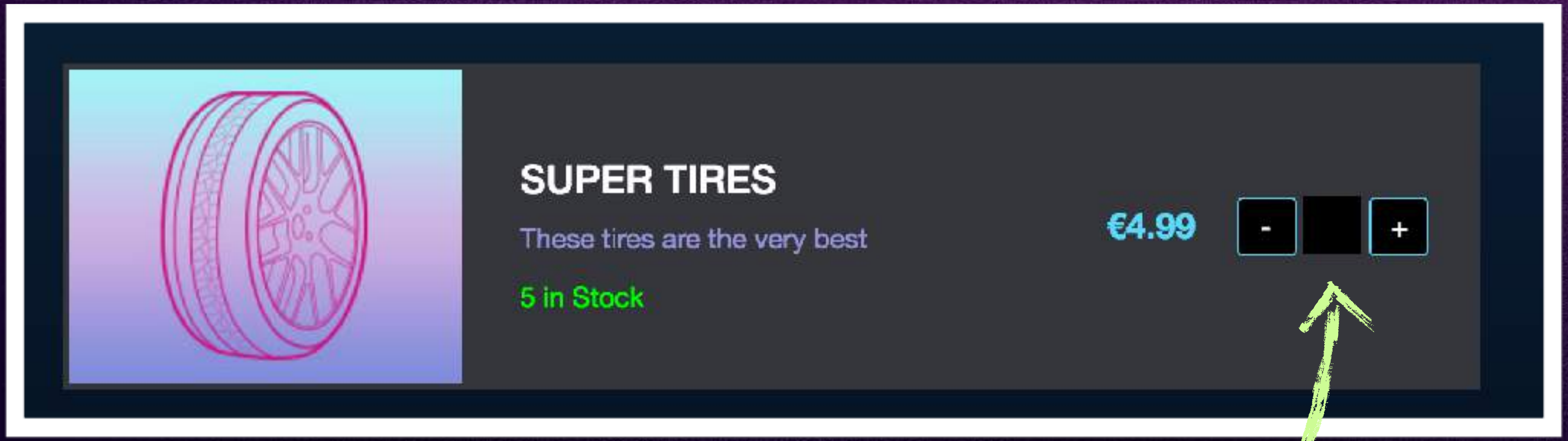
How can we bind properties from our component to our HTML, but also listen for events and keep things in sync?

Adding an Input Field

How can we allow for the user input of the quantity?

car-parts.component.html HTML

```html
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
     <button class="decrease" (click)="downQuantity(carPart)">-</button>
     <input class="number" type="text">
     <button class="increase" (click)="upQuantity(carPart)">+</button>
```



We should be able to adjust the quantity by typing into this field or by using the buttons.

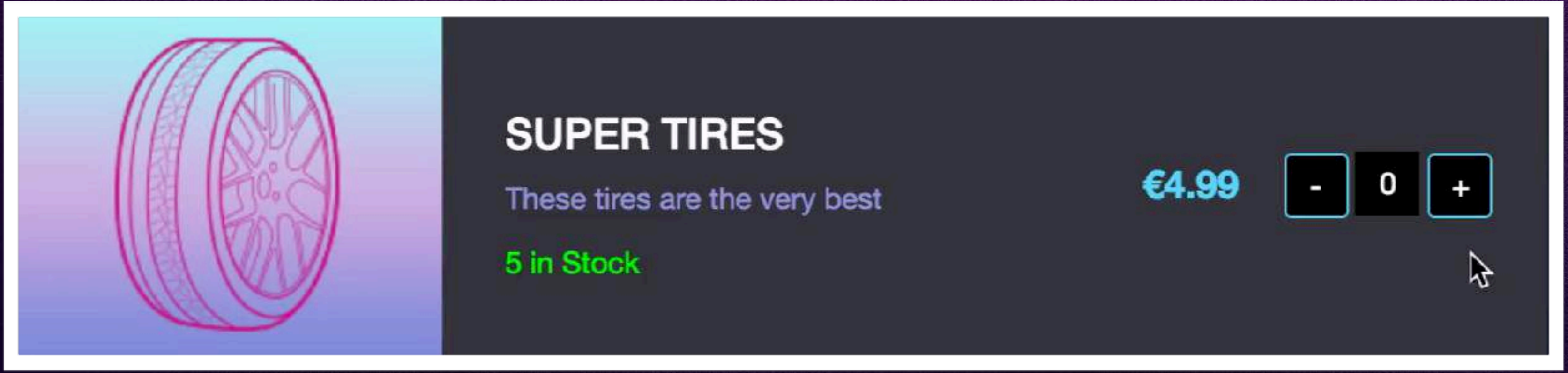
ACCELERATING THROUGH ANGULAR 2

Using Property Binding

The first thing we might try is to use property binding to bind the value to the quantity.

car-parts.component.html HTML

```html
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
     <button class="decrease" (click)="downQuantity(carPart)">-</button>
     <input class="number" type="text" [value]="carPart.quantity">
     <button class="increase" (click)="upQuantity(carPart)">+</button>
```


SUPER TIRES

These tires are the very best

5 in Stock

€4.99

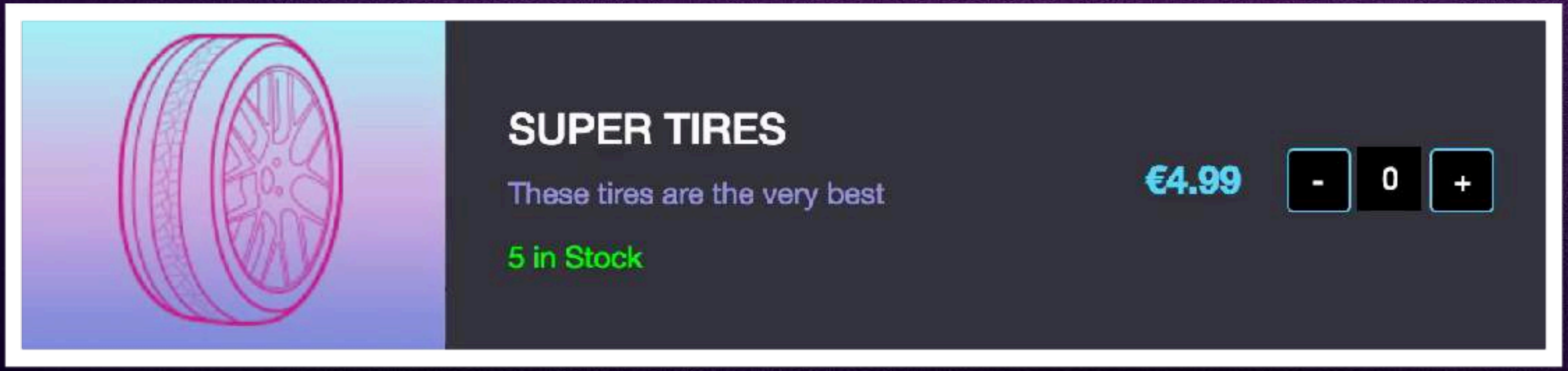
This gives us our quantity value in our input box, but only in one direction: from our component property to our input value.

Using Event Binding

We need to listen for the input event on our input box.

car-parts.component.html HTML

```html
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
     <button class="decrease" (click)="downQuantity(carPart)">-</button>
     <input class="number" type="text" [value]="carPart.quantity"
                                       (input)="carPart.quantity = $event.target.value">
     <button class="increase" (click)="upQuantity(carPart)">+</button>
```



SUPER TIRES

These tires are the very best

5 in Stock

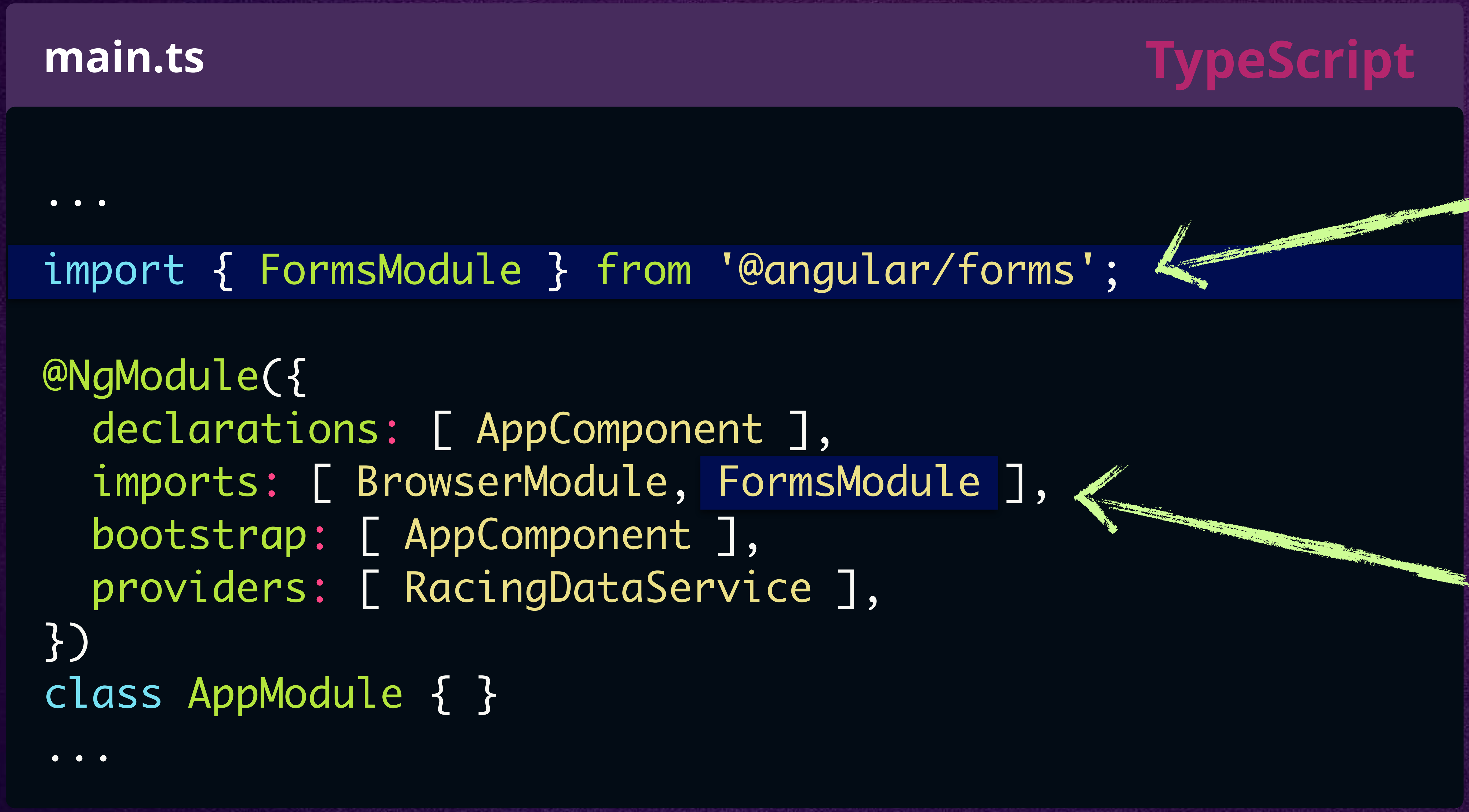
€4.99

Information is flowing two ways.

This works, but there's another way.

Importing the FormsModule

Let's import the FormsModule **to get additional forms functionality into our codebase.**



main.ts TypeScript

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

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule, FormsModule ],
  bootstrap: [ AppComponent ],
  providers: [ RacingDataService ],
})
class AppModule { }
...
```

Import FormsModule

Make form-specific functionality available to our whole app

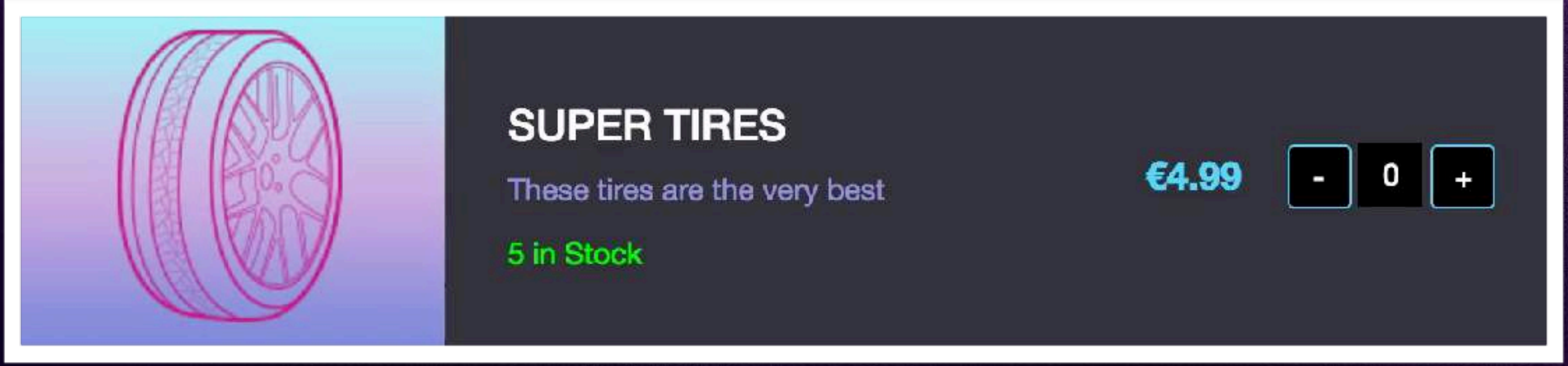
Using ngModel

ngModel **allows us to have one command to express two-way data binding.**

car-parts.component.html HTML

```
<td class="price">{{carPart.price | currency:'EUR':true }}</td>
<td>
  <div class="select-quantity">
     <button class="decrease" (click)="downQuantity(carPart)">-</button>
     <input class="number" type="text" [(ngModel)]="carPart.quantity" >
     <button class="increase" (click)="upQuantity(carPart)">+</button>
```

Notice that we're using both brackets and parentheses. [()]



SUPER TIRES

These tires are the very best

5 in Stock

This syntax is sometimes called banana in a box — can you see why?

The ngModel **Syntax**

When we use the ngModel **syntax, we can only set it equal to a data bound property.**

```
[(ngModel)]="<must be data property>"
```

We will mostly use this for form fields.

These are component properties:

```
[(ngModel)]="user.age"
```

✓

```
[(ngModel)]="firstName"
```

✓

This will error out:

```
[(ngModel)]="fullName()"
```

✗

What’d We Learn?

- The `[(ngModel)]` syntax allows us to specify a component property that will use two-way binding.
- Two-way binding means that if the component property is modified inside the component (JavaScript) or inside our web page (HTML), it will stay in sync.

ACCELERATING THROUGH
ANGULAR 2