DESARROLLADOR SOFTWARE – TALLER ANGULAR

FECHA- [19-20] - 2022 **Horario: 15:00 a 20:30 h**

Juan Antonio Herrerías Berbel

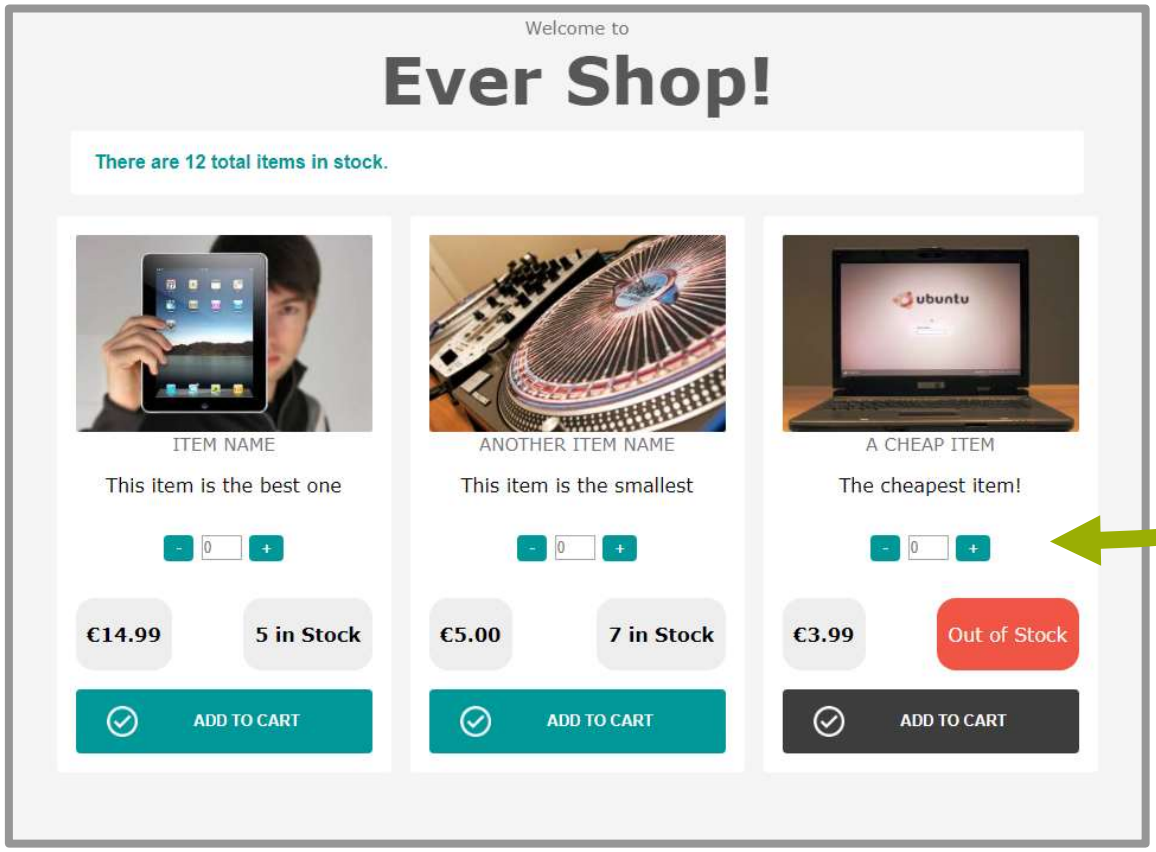
Índice

Herre - jaherrerias@gmail.com

— 06

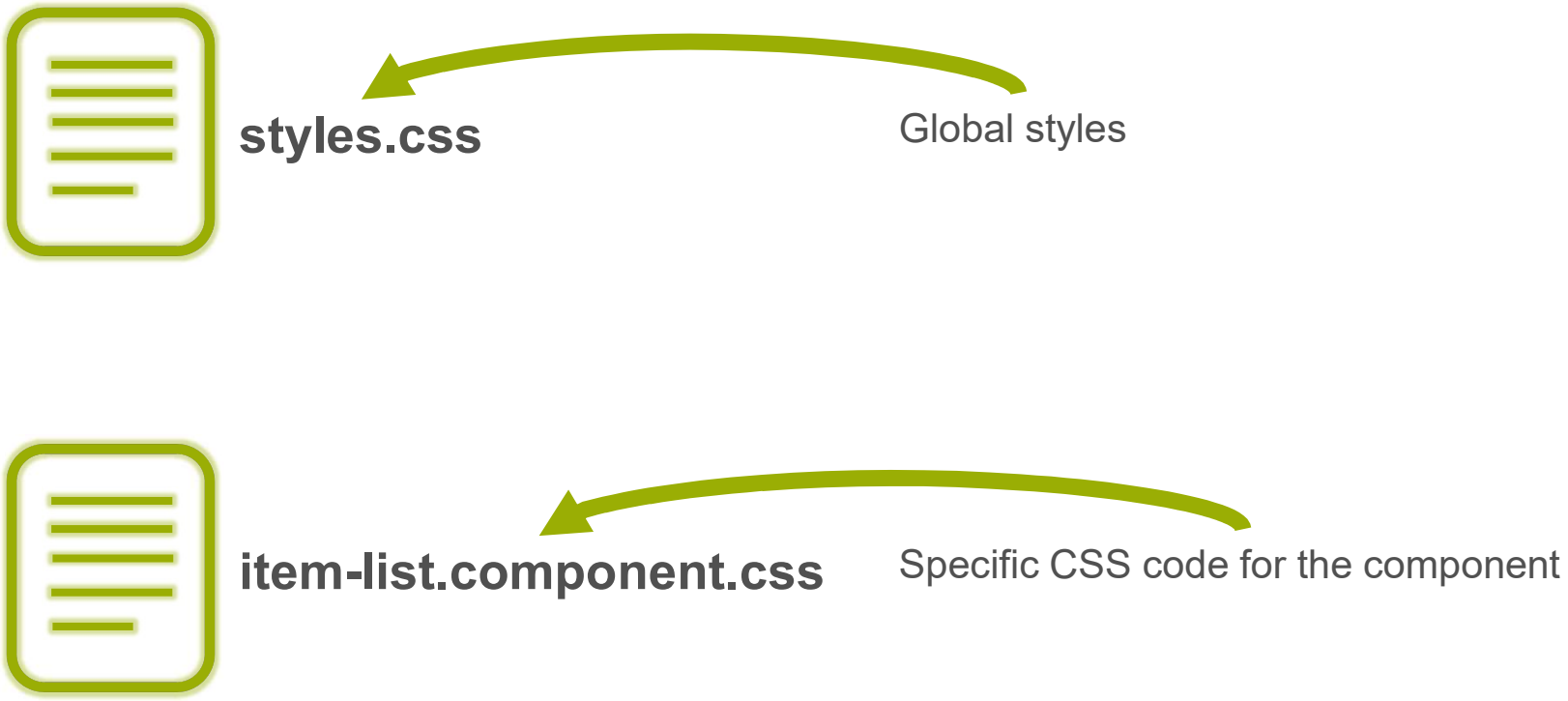
Property & Class Binding

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.



We'll be adding the images and item quantity after styling our code.

We can add global styles to *styles.css* file and the other css code to the specific css component file



HTML may be changed too if we use some specific CSS class

Our Current App With Styles

Better design, but how we bring images in?

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

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

item-list.component.html

```
<div class="total-stock">
  <p>There are {{totalItems()}} total items in stock.</p>
</div>

<div class="flex">
  <section *ngFor="let item of myItems">
      <h2>{{item.name | uppercase}}</h2>
      <p>{{item.description}}</p>
      <aside>
        <ul>
          <li>
            <p>{{item.price | currency:'EUR':true}}</p>
          </li>
          <li *ngIf="item.stock > 0">
            <p>{{item.stock}} in Stock</p>
          </li>
          <li *ngIf="item.stock === 0" class="out-stock">
            <p>Out of Stock</p>
          </li>
        </ul>
      </aside>
  </section>
</div>
```

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.

item.model.ts

```ts
export class Item {
    id: number;
    name: string;
    description: string;
    stock: number;
    price: number;
    image: string;
}
```

mocks.ts

```ts
import { Item } from './item.model';

export const ITEMS: Item[] = [{
    'id': 1,
    'name': 'Item name',
    'description': 'This item is the best one',
    'stock': 5,
    'price': 14.99,
    'image': 'headphone.jpg'
  },
  {
    'id': 2,
    'name': 'Another Item name',
    'description': 'This item is the smallest',
    'stock': 7,
    'price': 5,
    'image': 'gamepad.png'
  },
  {
    'id': 3,
    'name': 'A cheap Item',
    'description': 'The cheapest item!',
    'stock': 0,
    'price': 3.99,
    'image': 'raspberry.jpg'
  }];
```

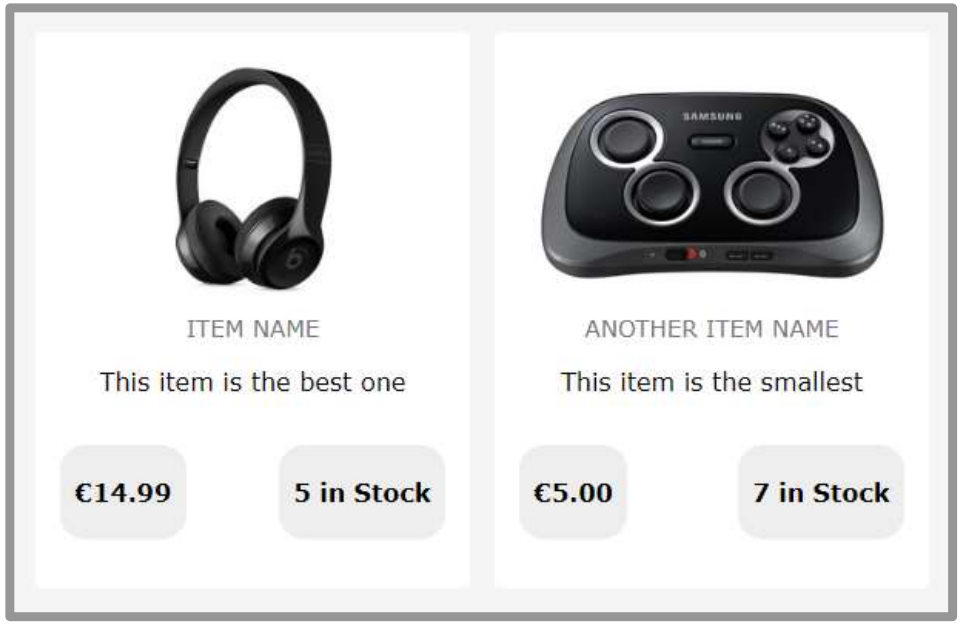
Adding an Image to Our Template

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

item-list.component.html

```
<div class="total-stock">
  <p>There are {{totalItems()}} total items in stock.</p>
</div>

<div class="flex">
  <section *ngFor="let item of myItems">
    <img src="{{'../../assets/'+item.image}}" />
    <h2>{{item.name | uppercase}}</h2>
    <p>{{item.description}}</p>
    <aside>
      <ul>
        <li>
          <p>{{item.price | currency:'EUR':true}}</p>
        </li>
```



This would work just fine.

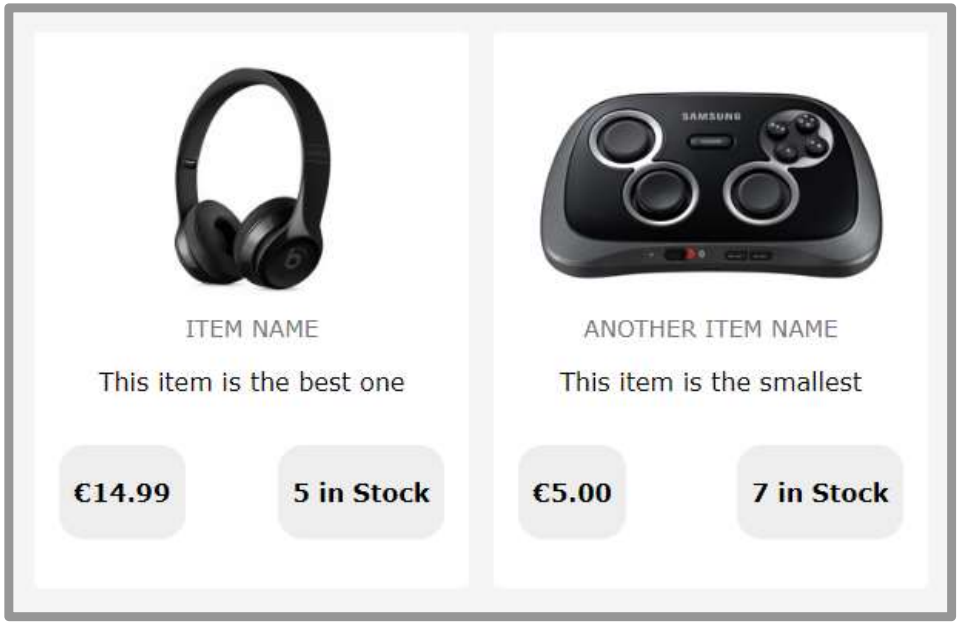
However, there's an alternative syntax we can use when we want to set DOM element property values.

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

item-list.component.html

```
<div class="total-stock">
  <p>There are {{totalItems()}} total items in stock.</p>
</div>

<div class="flex">
  <section *ngFor="let item of myItems">
      <img [src]="item.image" />
      <h2>{{item.name | uppercase}}</h2>
      <p>{{item.description}}</p>
      <aside>
        <ul>
          <li>
```



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 part of the DOM.

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

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

<button [disabled] ="isDisabled">Add to Cart</button>

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

If a item is marked as “selected,” we want to add a specific class to it.

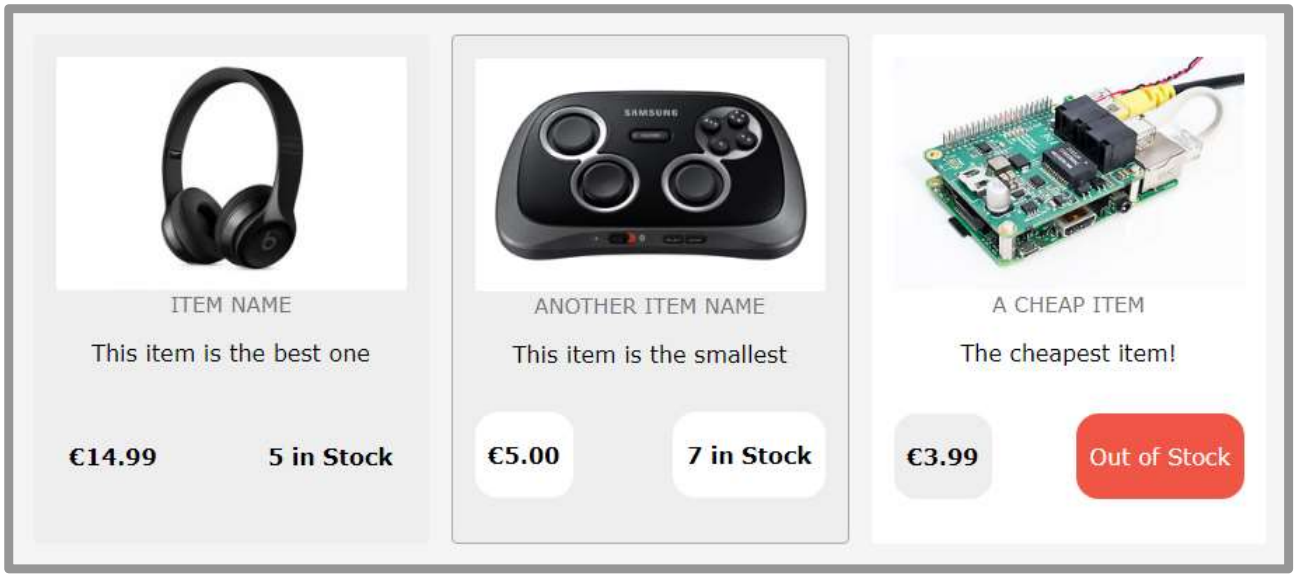
item-list.component.css

```css
.flex > section.selected {
  background: #eee;
  border: 1px solid #A8A8A8;
}

.flex ul {
  list-style-type: none;
  padding: 0;
}

.flex li {
  background: #eee;
  font-weight: 700;
  padding: 0.3em 0.6em;
  border-radius: 1em;
}

.flex li.selected {
  background: #fff;
}
```



How do we add functionality to sometimes add this *selected* class?

We need to add a new property to our item model and add mock data for it.

item.model.ts

```ts
export class Item {
    id: number;
    name: string;
    description: string;
    stock: number;
    price: number;
    image: string;
    selected: boolean;
}
```

mocks.ts

```ts
import { Item } from './item.model';

export const ITEMS: Item[] = [{
    'id': 1,
    'name': 'Item name',
    'description': 'This item is the best one',
    'stock': 5,
    'price': 14.99,
    'image':        assets/headphone.jpg',
    'selected': false
  },
  {
    'id': 2,
    'name': 'Another Item name',
    'description': 'This item is the smallest',
    'stock': 7,
    'price': 5,
    'image':        assets/gamepad.png',
    'selected': true
  },
```

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.

item-list.component.html

```
<div class="total-stock">
  <p>There are {{totalItems()}} total items in stock.</p>
</div>

<div class="flex">
  <section *ngFor="let item of myItems" [class.selected]="item.selected">
      <img [src]="item.image" />
      <h2>{{item.name | uppercase}}</h2>
      <p>{{item.description}}</p>
      <aside>
        <ul>
          <li [class.selected]="item.selected">
            <p>{{item.price | currency:'EUR':true}}</p>
          </li>
          <li *ngIf="item.stock > 0" [class.selected]="item.selected">
            <p>{{item.stock}} in Stock</p>
          </li>
          <li *ngIf="item.stock === 0" class="out-stock" [class.selected]="item.selected">
            <p>Out of Stock</p>
          </li>
        </ul>
    </aside>
  </section>
</div>
```

If **item.selected** is true, then the selected class is added. If **item.selected** is false, then the selected class is removed.

How Not to Use Class Binding

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

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

No!!!!
This will overwrite all classes.

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

This will only add/remove
the specific class.

- 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.

— 07

Event Binding

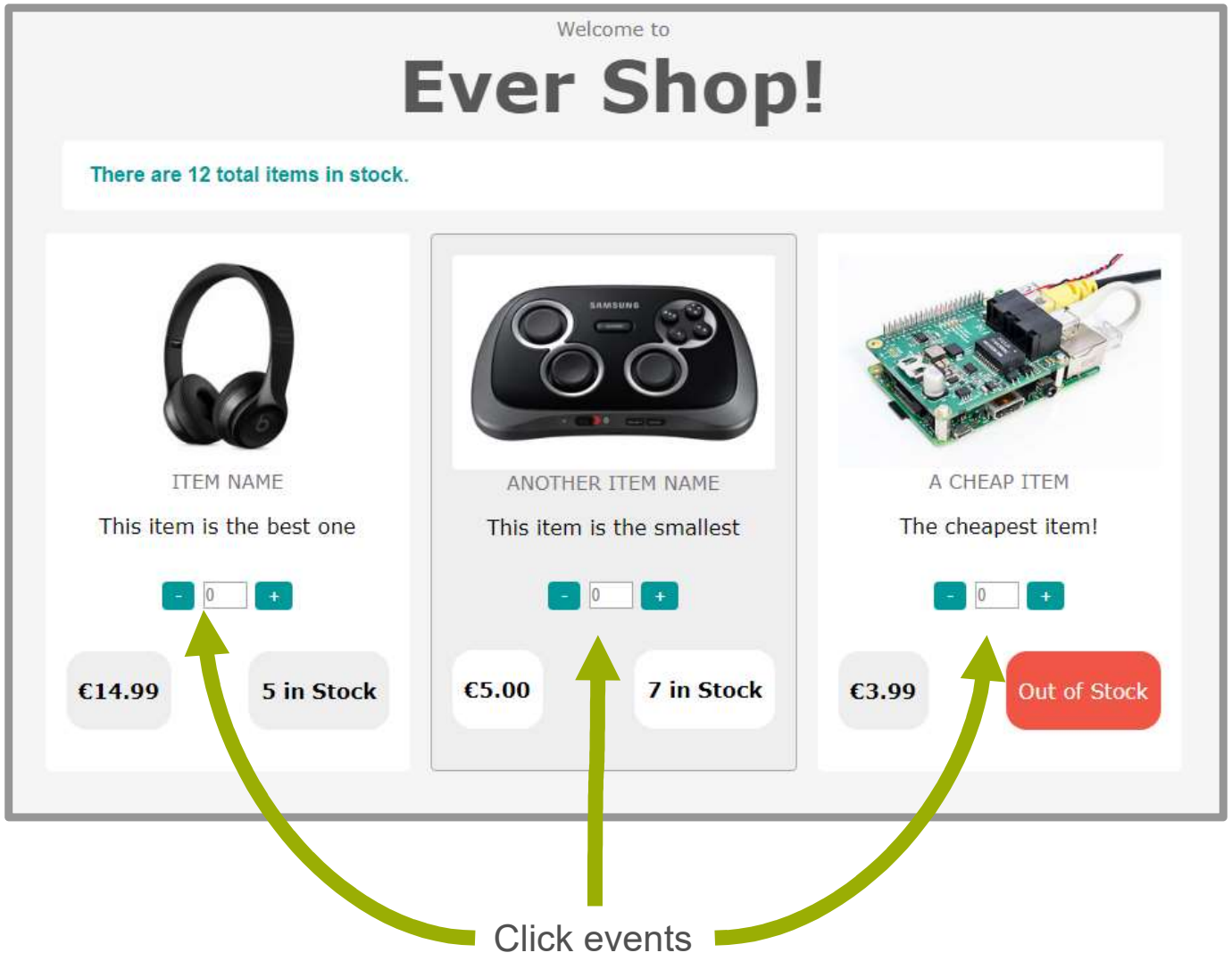
JavaScript to HTML

Property Binding
Class Binding

HTML to JavaScript

Event Binding

Like a mouse click, hover or key press


Welcome to
Ever Shop!
There are 12 total items in stock.
ITEM NAME
This item is the best one
- 0 +
€14.99
5 in Stock
ANOTHER ITEM NAME
This item is the smallest
- 0 +
€5.00
7 in Stock
A CHEAP ITEM
The cheapest item!
- 0 +
€3.99
Out of Stock
Click events

Adding a Quantity Property & Data

We need to add a new property to our item model and add mock data for it.

item.model.ts

```ts
export class Item {
    id: number;
    name: string;
    description: string;
    stock: number;
    price: number;
    image: string;
    selected: boolean;
    quantity: number;
}
```

mocks.ts

```ts
import { Item } from './item.model';

export const ITEMS: Item[] = [{
    'id': 1,
    'name': 'Item name',
    'description': 'This item is the best one',
    'stock': 5,
    'price': 14.99,
    'image': '../../assets/headphone.jpg',
    'selected': false,
    'quantity': 0
},
```

item-list.component.ts

```
. . . .
export class ItemListComponent implements OnInit {
            . . . .

            upQuantity() {
                        console.log('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.

item-list.component.html

```
<div class="panel-amount">
            <button class="btn-amount" (click)="upQuantity()">+</button>
</div>
```

Now let's use the *item.quantity* that we have on each item.

item-list.component.html

```
. . . .
<div class="panel-amount">
            <input type="number" maxlength="2" placeholder="0" class="amount" [value]="item.quantity"/>
            <button class="btn-amount" (click)="upQuantity(item)">+</button>
</div>
. . . .
```

We need to send the *current* item to our component.

But we need to make changes in our component in order to add only when we have in stock.

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

item-list.component.ts

```
. . . .
export class ItemListComponent implements OnInit {
  . . . .

  upQuantity(item: Item) {
    if (item.stock > 0) {
      item.quantity++;
      item.stock--;
    }
  }

}
```

Increment quantity

Decrement stock

item-list.component.ts

```
. . . .
export class ItemListComponent implements OnInit {
  . . . .
  downQuantity(item: Item) {
    if( item.quantity > 0 ) {
      item.quantity--;
      item.stock++;
    }
  }
}
```

We decrease the quantity but not below zero.

item-list.component.html

```
. . . .
<div class="panel-amount">
  <button class="btn-amount" (click)="downQuantity(item)">-</button>
  <input type="number" maxlength="2" placeholder="0" class="amount" [value]="item.quantity"/>
  <button class="btn-amount" (click)="upQuantity(item)">+</button>
</div>
. . . .
```

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

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

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

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

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

```
<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);
}
```

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.

- 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.

— 08

Two-way Binding

JavaScript to HTML

Property Binding
Class Binding

HTML to JavaScript

Event Binding

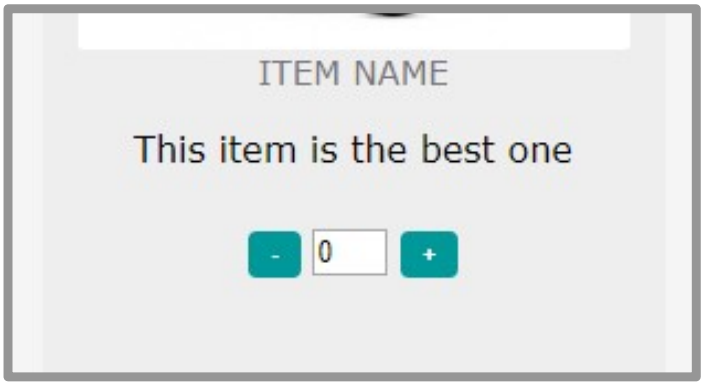
Both Ways

Like a text box, that should stay in sync

Checking Out our Input Field Using Property Binding

item-list.component.html

```
. . . .
<div class="panel-amount">
  <button class="btn-amount" (click)="downQuantity(item)">-</button>
  <input type="number" maxlength="2" placeholder="0" class="amount" [value]="item.quantity"/>
  <button class="btn-amount" (click)="upQuantity(item)">+</button>
</div>
. . . .
```



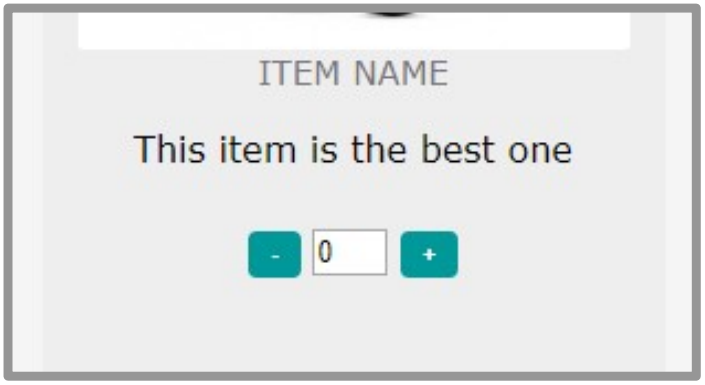
The first thing we tried was to use property binding to bind the value to the quantity.

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

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

item-list.component.html

```
. . . .
<div class="panel-amount">
  <button class="btn-amount" (click)="downQuantity(item)">-</button>
  <input type="number" maxlength="2" placeholder="0" class="amount" [value]="item.quantity“
                                    (input)="item.quantity = $event.target.value"/>
  <button class="btn-amount" (click)="upQuantity(item)">+</button>
</div>
. . . .
```



Now the information is flowing two ways.

But it’s really awful and there’s another way, the “Angular way”

Importing the FormsModule

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

app.module.ts

```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { ItemListComponent } from './item-list/item-list.component';

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

Import FormsModule from Angular core

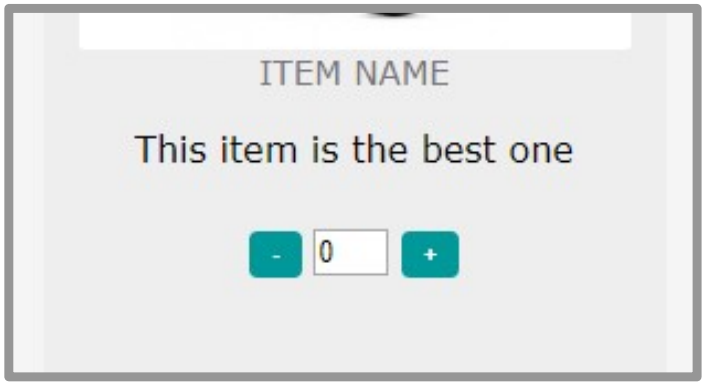
Make form-specific functionality available to our whole app

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

item-list.component.html

```
. . . .
<div class="panel-amount">
  <button class="btn-amount" (click)="downQuantity(item)">-</button>
  <input type="number" maxlength="2" placeholder="0" class="amount“ [(ngModel)]="item.quantity“/>
  <button class="btn-amount" (click)="upQuantity(item)">+</button>
</div>. . . .
```

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



This syntax is sometimes called "banana in a box"

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.

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

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

These are component properties uses in the right way

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

This will error out

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.

- Controlar los cambios que se hagan en el input de quantity y que haga el mismo funcionamiento que los botones de incremento y decremento.

- Controlar qué ocurre cuando se introducen valores mayores o menores que el stock.

- Marcar nuestro atributo de selected como true cuando hagamos un mouseover por el componente.

EOI

Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

ESTRATEGIA DE
EMPRENDIMIENTO Y
EMPLEO JOVEN
garantía juvenil

Gracias.