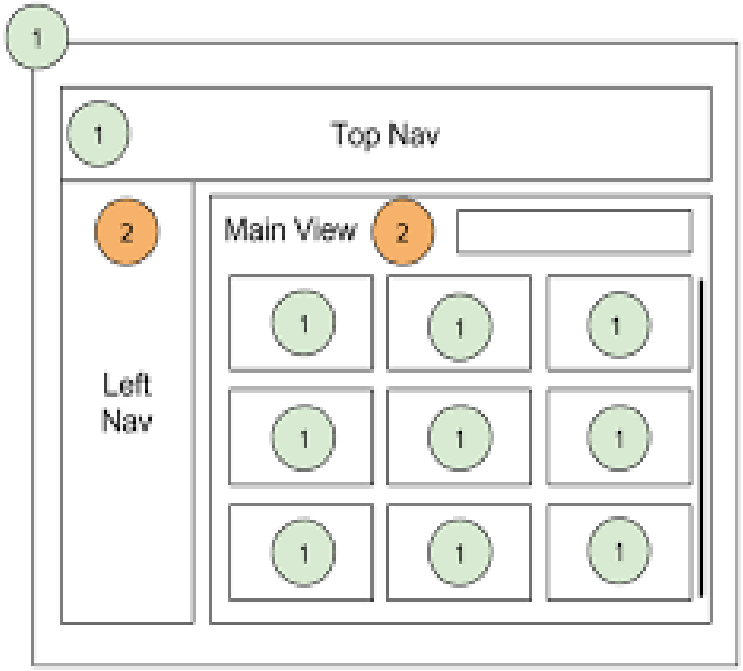
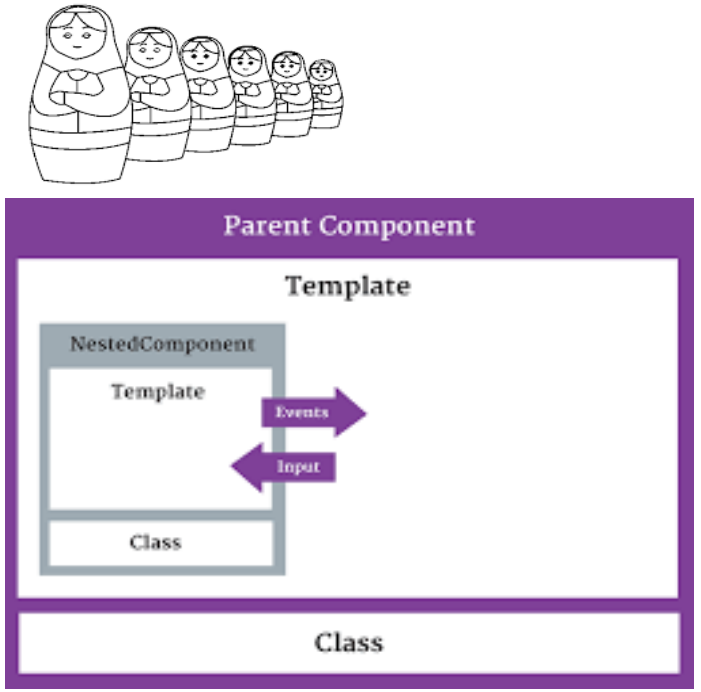
netmind
a bts company

Angular 14 - 08

Nested Components and inter-component communication

Structure and operation

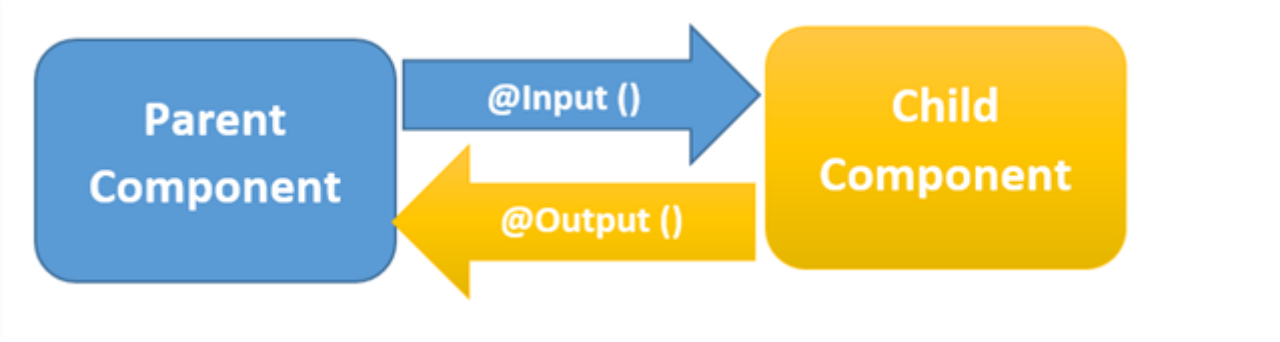




In Angular, components work like matryoshkas (Russian dolls): they use the container/content model.

Recommended situations

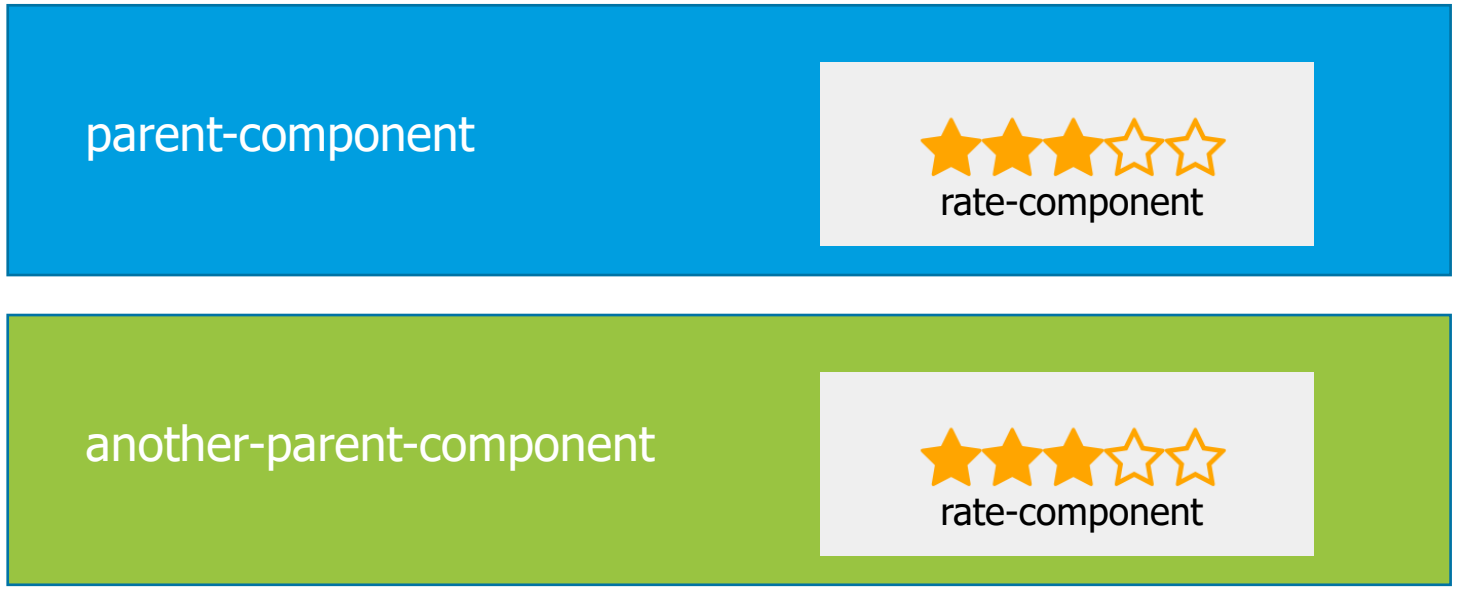
- It is recommended to declare a component as nested (nestable) in the following cases:
  - Your template only handles a fragment of a larger view.
  - It has a selector, so it can be used as a directive.
  - Optionally, it communicates with its container.
  - Nested components communicate with the outside using the **@input** and **@output** decorators.



Structure and operations

- For example, if we want to build a component to rate products with stars:
  - The rating component will have to receive an input parameter (the number) and transform that number into the corresponding stars.
  - In turn, we want that when the user clicks on the component, an event is emitted that notifies the possible changes.

- If we do it as another nested component, the component can be reused in various parts of the application.



The child component

- The HTML template would look something like this:

```
<div class="stars" [title]="rating">
    <div>
        <span [class]="rating>=1?'fill':''" (click)="starC1ick(1)"><i class="bi bi-star"></i></span>
        <span [class]="rating>=2?'fill':''" (click)="starC1ick(2)"><i class="bi bi-star"></i></span>
        <span [class]="rating>=3?'fill':''" (click)="starC1ick(3)"><i class="bi bi-star"></i></span>
        <span [class]="rating>=4?'fill':''" (click)="starC1ick(4)"><i class="bi bi-star"></i></span>
        <span [class]="rating>=5?'fill':''" (click)="starC1ick(5)"><i class="bi bi-star"></i></span>
    </div>

</div>
```

- Note that here we use bootstrap and boostrap icons

🔗 **Install bootstrap**

```
npm i bootstrap
npm i bootstrap-icons
```

🔗 **add to scss**

```
@import "~bootstrap/scss/bootstrap";
@import "~bootstrap-icons/font/bootstrap-icons";
...
```

The child component

- The content of the nested component will be similar to the previous ones, with the references to the UI files, plus their definitions.
- The component additionally needs to reference the **OnChanges** event mechanism , to respond to component changes.
    - We have to **import OnChanges** (for detection of changes in values input)
    - **Implement the ngOnChanges** method that takes care of the value.numeric => width.visual conversion .

```
import { Component, OnChanges} from '@angular/core';

@Component({
  selector: 'app-stars',
  templateUrl: './stars.component.html',
  styleUrls: ['./stars.component.scss'],
})
export class StarsComponent implements OnChanges {

  rating: number = 3;
  startC1ick = (rate:number) => {
    console.log('star clicked...',rate);
  };
  ngOnChanges(): void {
    console.log('ngOnChanges:');

  }
}
```

The parent component

- In turn, the outer template should use the nested component's selector:

```
<tr *ngFor="let product of products">
    <td>{{product.name}}</td>
    <td>{{product.code | lowercase}}</td>
    <td>
        <img *ngIf="show_imgs" [src]="'./assets/imgs/'+product.image"
                [title]="product.name | uppercase" [style.maxWidth.px]="img_width"
            [style.maxHeight.px]="img_height" />
    </td>
    <td>{{product.date}}</td>
    <td>{{product.price | currency:'USD':true:'1.2-2'}}</td>
    <td>
        <app-stars [rating]="product.stars"></app-stars>
    </td>
</tr>
```

- With this, we should see the rating converted to stars, but the component does not receive data or emit to the outside.
    - In order for the component to talk to the outside, we need to mark the properties (or events) of the component with **Communication Decorators**.

Communication decorator: @Input

- Communication Decorators allow us to mark properties with metadata that allow exposing their content to the outside and/or define which properties are considered input values for the component.
- To do this we decorate the elements whose value is received from the outside using the **@Input** decorator

```
@Component()
export class StarsComponent implements OnChanges {

  @Input() rating: number = 3;
  ...
}
```

- Once this is done, we can pass data to that element using "property binding" in the container template.

```
...
                    <app-stars [rating]="product.stars"></app-stars>
...
```

- That way the value of the rating is received by the nested component in its "rating" property.

Communication decorator: @Output

- If we want the nested component to communicate information to the outside, we must use the **@Output** decorator.
  - The way to declare this situation in the component is by defining an event, represented by the **EventEmitter** class.
  - EventEmitter allows the use of generics, so we can define it using that syntax and indicate the type of data we want to pass to the container.
  - In this case, it is enough for us to communicate it to the outside through a chain that informs the UI which element has been selected (clicked), so that it responds as appropriate.
  - In the code we will add the definition of the event in this way:

```
...
@Output() star_clicked: EventEmitter<number> = new EventEmitter<number>();
...
```

Communication decorators (@Input, @Output)

- Additionally, we will need to import the Output and EventEmitter definitions into the nested component.
- Finally, we will add the onClick event to indicate the behavior we want at runtime.

```
import {
  ...
  EventEmitter,
  Output
} from '@angular/core';
...
@Output() star_clicked: EventEmitter<number> = new EventEmitter<number>();

  starC1ick = (rate: number) => {
    console.log('star clicked...', rate);
    this.star_clicked.emit(rate);
  };

...
```

Reception of events in the parent

- In the outer user interface (the listing), we will refer to this (inner) event, by:

```
<app-stars [rating]="product.stars" (star_clicked)="onRatingClicked($event,product.code)"></app-stars>
```

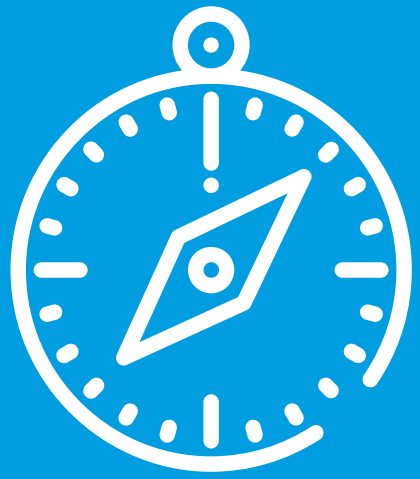
- And in its content, onRatingClicked will also be defined:

```
  ...
  onRatingClicked=(new_rating:number,code:string)=>{
    console.log('onRatingClicked:',new_rating, code);
    // process the input
  }
...
```

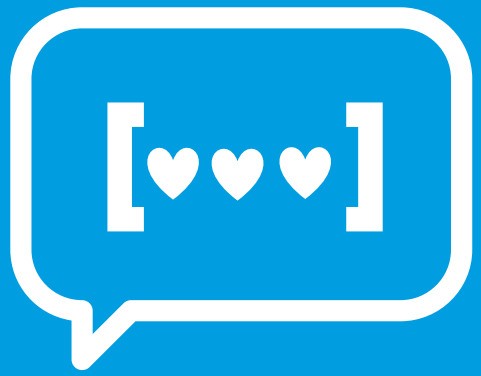
Let's put it into practice: Tasks/Projects App

Add a nested component to each row of the task list (a button).

This component should display a button. When clicked it will delete the corresponding task.



Next steps



We would like to know your opinion!

Please, let us know what you think about the content.
From Netmind we want to say thank you, we appreciate time and effort you have taking in answering all of that is important in order to improve our training plans so that you will always be satisfied with having chosen us
quality@netmind.es

netmind
a bts company

Thanks!

Follow us: