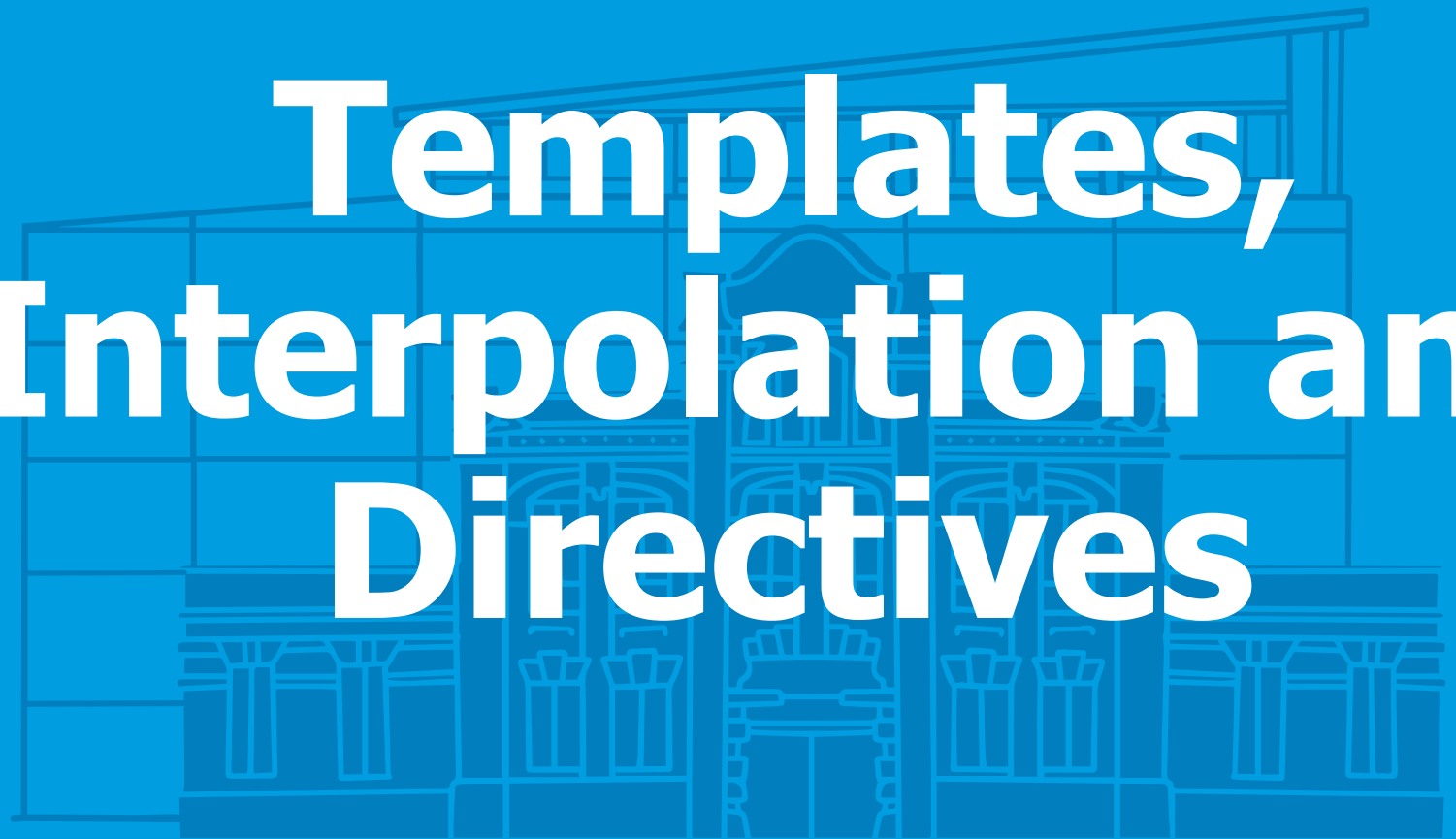
netmind
a bts company

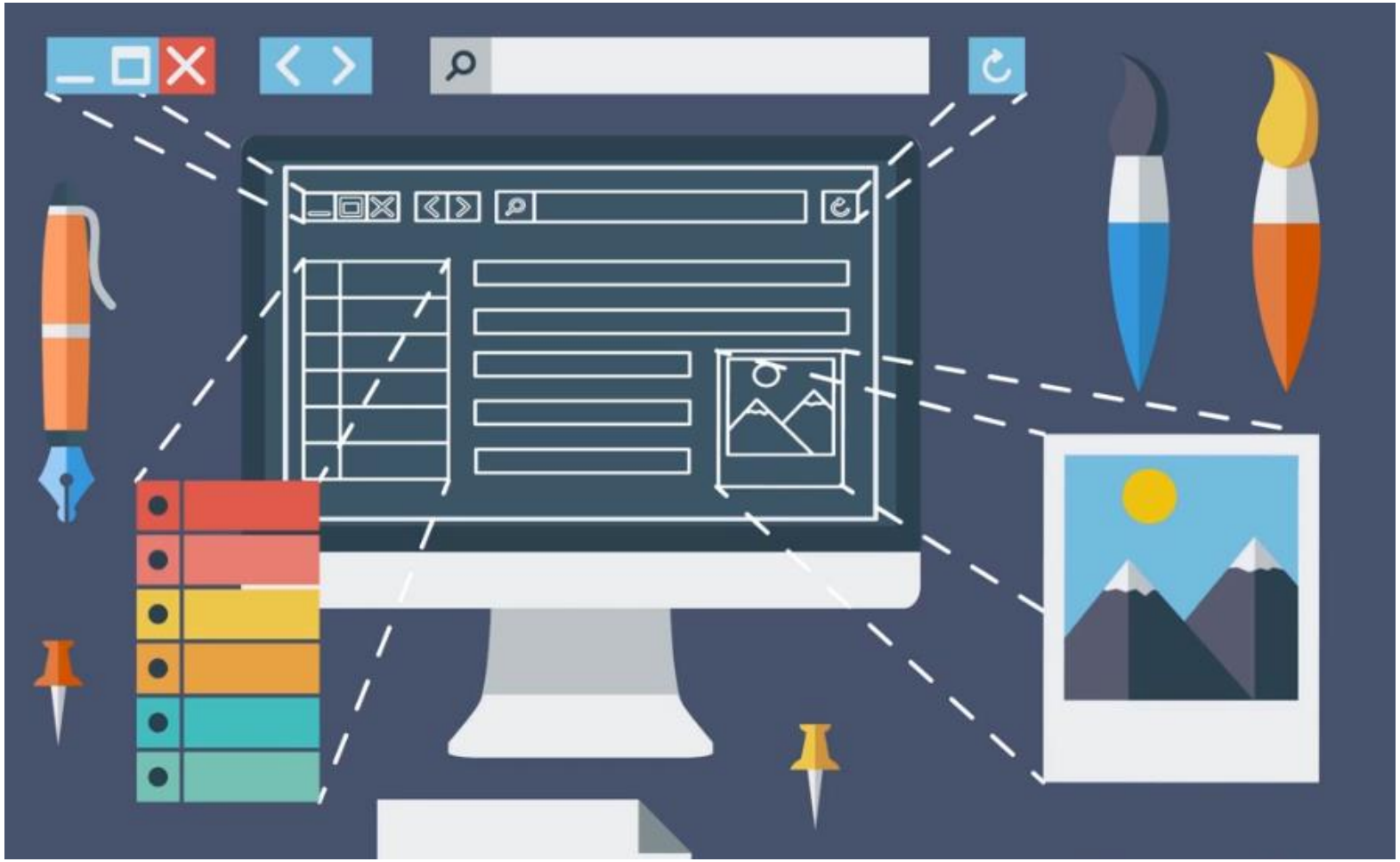
Angular 14 - 06

Templates, Interpolation and Directives



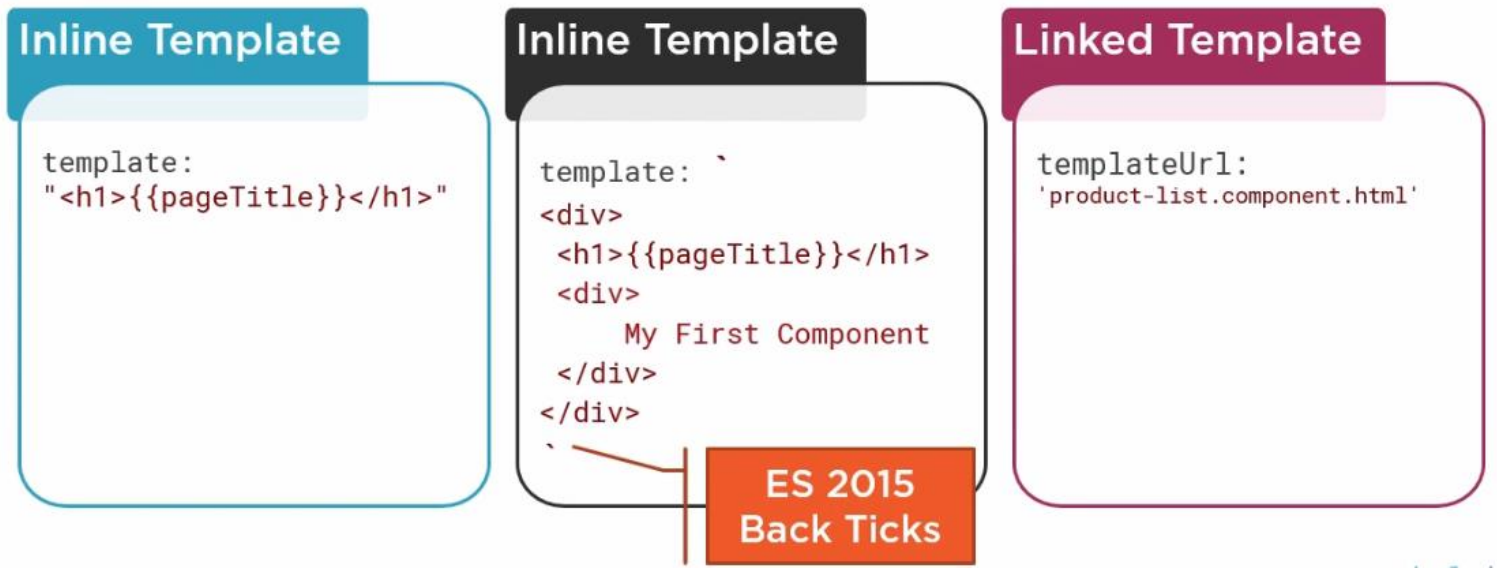
Composition of a view

The app will be a set of UI elements, which will be integrated like a puzzle



Template types

- We can distinguish 3 ways of creating templates:



- Although the first two are useful, in real projects we will tend to use the 3rd option.

Template interpolation

- If we add a property to the component class, we can **interpolate** its content from the template using **mustache** syntax .
- This is the one-way write-only binding of view models.

```
// component class
...
export class AppComponent {
  title = 'my-new-app';
}
```

```
// tmmplate
...
<span>{{ title }} app is running!</span>
...
```

Component selector

- By convention, a directory layout is used that follows the DDD principles (Domain Driven Design):
- Each main section of the application has its own directory, within the app directory
- We define a selector (product-list) and a template.

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

@Component({
  selector: 'app-products-list',
  templateUrl: './products-list.component.html',
  styleUrls: ['./products-list.component.scss']
})
export class ProductsListComponent implements OnInit {

}
```

Linking components

- In order for our component and its template to be recognized by the component that hosts them, we use their selector on the parent component.

```
<section>
  <h1>{{title}}</h1>
  <app-products-list></app-products-list>
</section>
```

Built-in directives

- In addition to creating our own directives, Angular includes a set of built-in directives.
  - https://angular.io/guide/built-in-directives
- Structural directives modify the structure of the DOM /View:
  - https://angular.io/guide/structural-directives

```
@Component(...)
export class ProductsListComponent {
  show_text = false;
  text = 'this is a text';
}
```

```
    <div *ngIf="show_text">{{text}}</div>
```

- Syntactically, they are distinguished from the rest by the presence of the asterisk (*) in front of the directive definition.

Built-in directives

- We can test this by adding that directive to a table, since at the moment, it doesn't contain any elements.
- The directive has to refer to some accessible object in the component, so we create a products input , indicating that it is an array of type any .

```
@Component(...)
export class ProductsListComponent {
  show_text = true;
  text = 'Available products';

  no_products: string = 'No products at the moment...';
  products: any[] | null = [];
}
```

- And to the <table> tag , we add a *ngIf directive, so that it only shows the table if there is a product definition and it is not empty.

```
<table *ngIf="products" class="products">... </table>
```

- If you view after this, you will see that the table does not appear.

Built-in directives

- We can do a first data presentation test, which looks like the final result, by adding elements to that product definition.
- To do this, we simply add a couple of elements to the component in the definition
- You just have to remember the names of the fields afterwards, to include them exactly as they are in the view.

```
@Component(...)
export class ProductsListComponent {
   ...
   products: any[] | null = [
    {
      name: 'Leaf Rake',
      code: 'GUN-0611',
      date: 'March 19, 2916',
      price: 19.95,
      stars: 3.2,
    },
    {
      name: 'Garden Cart',
      code: 'GUN-0023',
      date: 'March 21, 2916',
      price: 32.99,
      stars: 4.2,
    },
  ];

}
```

Built-in directives

- We can also use the *ngFor directive to generate repetition of directives
- For example, we add a global row entry to the table and tell it to iterate through the collection and create as many elements as the data array has.

```
<section>
    <h3 *ngIf="show_text">{{text}}</h3>
    <div *ngIf="!products" class="red">{{no_products}}</div>
    <table *ngIf="products" class="products">
        <thead>
            <tr>
                <th>Product</th>
                <th>Code</th>
                <th>Date</th>
                <th>Price</th>
                <th>Stars</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let product of products">
                <td>{{product.name}}</td>
                <td>{{product.code}}</td>
                <td>{{product.date}}</td>
                <td>{{product.price}}</td>
                <td>{{product.stars}}</td>
            </tr>
        </tbody>
    </table>
</section>
```

Built-in directives

- With all those changes, the final output of the list would only have a couple of elements, but we can already get an idea of the final result.
- It should look similar to the following:

Products store

Available products

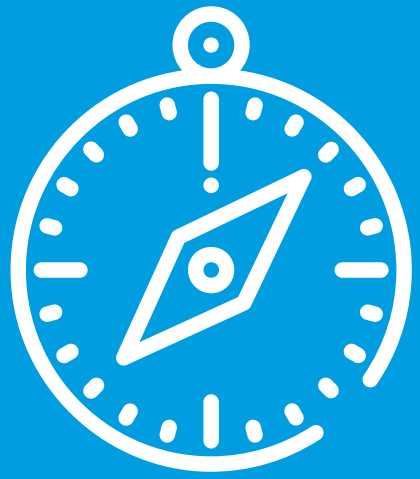
Product	Code	Date	Price	Stars
Leaf Rake	GUN-0611	March 19, 2916	19.95	3.2
Garden Cart	GUN-0023	March 21, 2916	32.99	4.2

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

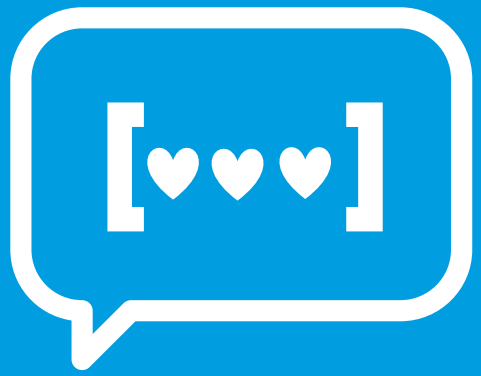
Create a template for tasks and projects lists components:

1. Task Model
    - tid:number
    - description:string
    - time:number
    - project:number
2. Project Model
    - pid:number
    - name:string
    - date:Date

Make the two lists display on one page using tables and HTML lists



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: