LAB4 - Displaying list of tasks

In this lab, we will get to use built-in Angular structural directives like ngIf for conditional rendering and ngFor for loops. We will also learn how to use attribute directives like ngClass and ngStyle to style and enhance the UI.

Showing the list of tasks

In this section we will render the list of tasks,

1. Open the starter project (from the previous lab)
2. Open the `app.component.html` file , it contains only the `<app-task-creator>` component
3. Add an `ul` element to show the list of tasks,

```
<ul>
    //items over here
</ul>
```

4. Now using `*ngFor` directive, loop over the `tasks` field, and for each task add a `li` item showing the `description` field

```
 <ul>
     <li *ngFor="let task of tasks">
         {{task.description}}
     </li>
 <ul>
```

5. In order to use `Priority` enum fields in the HTML template, you'll need to expose it throught a getter method in the component TS class, like the following:

```
 get Priority() {
     return Priority
 }
```

6. Tasks with high priority are important, for these latters we want to see also whether they're completed or not. You can do that using `*ngIf` to conditionaly show or hide the `completed`

information

```
<li *ngFor="let task of tasks">
    {{task.description}} <span *ngIf="task.priority==Priority.High">{{task.comple
</li>
```

Extracting the tasks list component

For a better code structure, we will extract the list of tasks into a new component.

1. Using the Angular CLI, generate a new component: `TasksList`

```
ng generate component tasks-list
```

2. Move the code that shows the list of tasks into the new component.

3. Now, the child component (TasksListComponent) needs the `tasks` value from the parent component, pass it throught `@Input` decorator

```
@Input() tasks: Task[] = []
```

4. Don't forget to update `app.component.html` to display the tasks list component

```
<app-tasks-list  [tasks]="tasks"  ></app-tasks-list>
```

Adding some styles

In this section, we will be using **ngClass** and **ngStyle** to style the tasks list component

1. We want to give tasks with different priorities different colors, green for low, grey for normal and red for high. To acheive that create three CSS classes in `tasks-list.component.css`

```
.low {
    color: green;
}
.normal {
    color: grey;
}
.high {
```

```
    color: red;
}
```

2. Now bind those classes with the correponding priorities using `ngClass` directive

```
<li  *ngFor="let  task  of  tasks"
        [ngClass]="{'low': task.priority==Priority.Low,
                    'normal': task.priority==Priority.Normal,
                    'high': task.priority==Priority.High }">
            //...
</li>
```

3. To add some margin around the list item and make the font a bit larger, you can use `ngStyle` directive

```
[ngStyle]="{'margin':'10px', 'font-size': '20px'}"
```

> Those styles are not complex and does not really require an ngStyle directive, we just used it for the purpose of the lab.

Custom directive

In this section, we will implement a custom attribute directive, in order to highlight high priority task.

1. In the `src/app` folder run the following command to generate a new directive called `important`

```
ng generate directive important
```

> You can delete the file `important.directive.spec.ts` which is dedicated for unit testing

2. Go the `important.directive.ts`, replace the selector `[appImportant]` by `[important]`

```
 @Directive({
     selector:  '[important]'
})
export  class  ImportantDirective {
    constructor() { }
}
```

3. Inject `ElementRef` to get the current element for which this directive is applied

```
 constructor(private element:ElementRef) { }
```

4. This directive should highlight important tasks, for example setting a text decoration, create a private method called `highlight` and implement it to change the style of the element, (you're free to apply the style of your choice)

```
private highlight() {
    this.element.nativeElement.style["text-decoration"] = 'underline'
}
```

5. This directive needs to be applied only to high priority tasks, therefore we need this information inside this directive to conditionaly apply this highlighting. Add an `@Input` for this directive :

```
@Input("important") isImportant:boolean = false
```

> Seting the alias of the input the same as the directive selector allows to send the value through the same attribute in the template

6. Implement the `OnInit` interface and call the `highlight` from the `ngOnInit` hook to perform actual changes

```
ngOnInit() {
    if (this.isImportant) {
        this.highlight()
    }
}
```

7. Go to `tasks-list.component.html` and apply the `important` directive to `li` elements

```
[important]="task.priority==Priority.High"
```