LAB3 - Use multiple Components

In the previous lab, we created the form directly in the app component, when the project start growing this structure won't be optimal, Angular is designed to hold a tree of components, each component having clear responsibilities. In this lab, we will be creating a new component and extracting the task form into this component.

Create a new component

1. We need another component to hold our task form, use the Angular CLI to generate a new component, call it `TaskCreatorComponent`

   ```
   ng generate task-creator
   ```

2. Once the component is created, move the content of `app.component.html` into the file `task-creator.component.html`
3. Move also the necessary properties from the app component class (`app.component.ts`) file

   > Keep the `tasks` property in `app.component.ts` file

4. Now the app component class should contains only the `tasks` property

Add the component to the UI

To show an Angular component, it must be part of the components tree starting from the root component (which is app component), to do that use the component `selector` (the property in the `@Component` decorator).

Head over the `app.component.html` and add the task creator component as en element

```
<app-task-creator></app-task-creator>
```

Setup parent-child communication

Now that we've extracted the form into a separate component, we have to setup communication between the parent component (`app-component`) and the child component (`task-creator`)

Currently, the default `priority` for the new task is hardcoded (As `Priority.Normal`) in the `newTask` property, we want to set it dynamically. To acheive that, you will be using `@Input` to send the default priority from the parent component (`AppComponent`)

1. Add a new field in the `TaskCreatorCreator` , call it `defaultPriority` and annotate it with `@Input` decorator :

```
@Input() defaultPriority: Priority = Priority.Normal
```

2. You can give this property an alias `default` to keep the code readable in the HTML content

```
@Input("default") defaultPriority: Priority = Priority.Normal
```

3. Update the `newTask` initialization and set it to the `defaultPriority`

```
newTask: Task = {
    id:  0,
    description:  "",
    completed:  false,
    priority:  this.defaultPriority
};
```

4. Add a default priority in the parent component (`AppComponent`), and send it to the child component (`TaskCreatorComponent`) using property binding like the following

```
//TS
defaultPriority: Priority = Priority.Normal
//HTML
<app-task-creator  [default]="defaultPriority"></app-task-creator>
```

Setup child-parent communication

When adding a task, the `addTask()` method previously added the new task into `tasks` property, now it have send the new task back to the parent. To acheive that, you will be using `@Output` with `EventEmitter` to emit an event once the task creator add a new task.

1. Add a new field in the `TaskCreatorCreator` , call it `onTaskCreated` and annotate it with `@Output` decorator :

```
@Output() onTaskCreated = new EventEmitter<Task>()
```

2. Update the `addTask` method to make it `emit` an event once it finished creating the task

```
addTask(task:Task) {
    //...implementation here
    this.onTaskCreated.emit(task)
}
```

3. Add a method `onTaskCreated` in the parent component that will respond to the emitted event from the child component, this method should add the received `task` into the list of `tasks`

```
onTaskCreated(task:Task) {
    this.tasks.push(task)
    console.log(tasks)
}
```

4. Now, you have to bind this method to the output field of the child component using event binding

```
<app-task-creator  [default]="defaultPriority"
                    (onTaskCreated)="onTaskCreated($event)">
    </app-task-creator>
```

> $event object contains the task object of type `Task` (according to `EventEmitter<Task>`)

Using lifecycle hooks

If you run the code like this, everything should work except one issue, if you change the value of the `defaultPriority` sent by the app component, you will notice it stays always to the **Normal** priority, we can explain this inconsistency by the following code:

```
@Input("default") defaultPriority: Priority = Priority.Normal
//...
newTask: Task = {
    //...
    priority:  this.defaultPriority
};
```

Here we set up the `priority` property of the `newTask` to the `defaultPriority` directly in the **statement assignement at the class level**, the issue is that at the moment of instanciating the component (calling the constructor) the inputs are not populated yet so it sticks to the default value (Priority.Normal). To resolve that issue you have to use inputs only after they have been populated, in other words after `ngOnChanges` lifecycle hook.

> Reminder: `ngOnInit` is called after `ngOnChanges` hook

You can use `ngOnInit` hook to initialie the `newTask` object:

1. Make sure that `TaskCreator` component implement `OnInit` interface

```
export  class  TaskCreatorComponent  implements  OnInit {
    //...
    ngOnInit(): void {
    }
}
```

2. Move the `newTask` initialization to the `ngOnInit` method

```
 ngOnInit(): void {
     this.newTask = {
         id:  0,
         description:  "",
         completed:  false,
         priority:  this.defaultPriority
     };
 }
```

Now, if you run again and change the inputs from the parent component the child component should respond to changes correctly.