YASH Technologies

Todo – App

Multiple Components

Ref : angular Hero App

Multiple Component Need

The AppComponent is doing everything that is expected, in the beginning it had shown the single todo. Slowly we have converted this AppComponent to demonstrate the functionality of the Master/Detail with both list of todos and todo details.
Slowly our App will have more features, we can not put all the code here in one Component.

Better design will be to break it up into sub-components, each focused on a specific task and workflow. Slowly AppComponent will become a simple shell that hosts those sub-components.

In this session you will be taking the first step in that direction by separating out the todo details in a separate component.

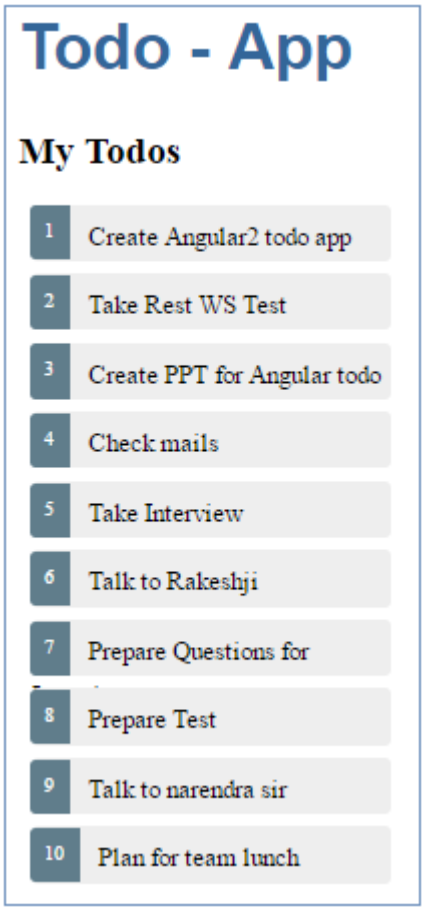
Before starting first run the app and verify that the last changes are reflected.

Better would be that you create a separate project and name it something you want, copy and paste the code from the last app.

Learning points

- You will learn how to create reusable component.
- You will learn how to make a component accept input.
- You will lean how to declare required application directives in an Angular module.
- You will know how to list the directives in the NgModule decorator’s declaration array.
- You will learn to bind the parent component to a child component.

Final Product at the End of this session



Keep the app running

Write below command in the command line

```
npm start
```

This command runs the TypeScript compiler in "watch mode", recompiling automatically when the code changes. The command simultaneously launches the app in a browser and refreshes the browser when the code changes.

You can keep building the Todo-App without pausing to recompile or refresh the browser.

Create a TodoDetailComponent

Create a file named as **todo-detail.component.ts** in app folder. Here we will create new component as TodoDetailComponent.

There is a naming convention for component.

STYLE 05-02

> **Do** use *dashed-case* or *kebab-case* for naming the element selectors of components.
>
> **Why?** Keeps the element names consistent with the specification for Custom Elements.

app/heroes/shared/hero-button/hero-button.component.ts

```
/* avoid */

@Component({
  selector: 'tohHeroButton',
  templateUrl: './hero-button.component.html'
})
export class HeroButtonComponent {}
```

Create a TodoDetailComponent

Create TodoDetailComponent

Naming convention while creating TodoDetailComponent

The component *class* name should be written in *upper camel case* and end in the word "Component". The todo detail component class is TodoDetailComponent.

The component *file* name should be spelled in *lower dash case*, each word separated by dashes, and end in .component.ts. The TodoDetailComponent class goes in the hero-detail.component.ts file.

Create a TodoDetailComponent

Write below code to create TodoDetailComponent

```
src/app/todo-detail.component.ts(TodoDetailComponent)
------------------------------------------------
import { Component } from '@angular/core';
@Component({
selector: 'todo-detail',
})
export class TodoDetailComponent{
}
```

To define the component , you need to import the **Component** symbol from '**@angular/core**'

@Component decorator provides the Angular metadata for the component. The CSS selector name, todo-selector, will match the element tag that identifies this component within a parent component's template. You will be adding <todo-detail></todo-detail> tag in the AppComponent template.

Always **export** the Component class, because you will be **importing** it somewhere else.

Todo detail template

To move the todo detail view to the TodoDetailComponent, cut the todo detail contents from the bottom of the AppComponent template and paste it into new template property in the @Component metadata.

The TodoDetailComponent has a todo, not a selected todo. Replace the word “selectedTodo” with the word “todo” everywhere in the template. When you are done the new template should look like the below one.

```
src/app/todo-detail.component.ts(TodoDetailComponent)
-----------------------------------------------
import { Component } from '@angular/core';
@Component({
selector: 'todo-detail',
template:`
<div *ngIf="todo">
<h2>{{todo.name}} - details!</h2>
<div><label>id: </label>{{todo.id}}</div>
<div>
<label>name: </label>
<input [(ngModel)]="todo.name" placeholder="add task">
</div>
</div>
`
})
```

Add the todo property

The TodoDetailComponent template binds to the component's todo property. Add that property to the TodoDetailComponent class.

```
src/app/todo-detail.component.ts(TodoDetailComponent)
-----------------------------------------------
export class TodoDetailComponent{
todo: Todo;
}
```

The todo property is an Instance of Todo, but we do not have any reference of Todo class, that is defined in app.component.ts file. Now there are two components that need the reference of Todo class. Angular recommends one class per file.

In the next slide look for the angular single responsibility principle (SRP) first, then we will be resolving this issue as well.

Angular SRP (Single Responsibility Principle)

Single responsibility

Apply the *single responsibility principle* (SRP) to all components, services, and other symbols. This helps make the app cleaner, easier to read and maintain, and more testable.

Rule of One

STYLE 01-01

Do define one thing, such as a service or component, per file.

Consider limiting files to 400 lines of code.

Why? One component per file makes it far easier to read, maintain, and avoid collisions with teams in source control.

Why? One component per file avoids hidden bugs that often arise when combining components in a file where they may share variables, create unwanted closures, or unwanted coupling with dependencies.

Why? A single component can be the default export for its file which facilitates lazy loading with the router.

The key is to make the code more reusable, easier to read, and less mistake prone.

Add the todo property

Move the Todo class from app.component.ts to its own todo.ts file.

src/app/todo.ts

```
src/app/todo.ts(Todo class)
-----------------------------------------------
export class Todo{
id:number;
name: string;
}
```

Now Todo class is its on todo.ts file, the AppComponent and the TodoDetailComponent have to import it. Add the below code in both app.component.ts and todo-detail.component.ts file.

```
src/app/app.component.ts | src/app/todo-detail.component.ts
-----------------------------------------------
import { Todo } from './todo';
```

Save the changes and look for bowser now.

todo property is an input property

Latter in this session, the parent AppComponent will tell the child TodoDetailComponent which todo to display by binding its seletedTodo to the todo property of the TodoDetailComponent. The binding will look like the below code.

<todo-detail [todo]="selectedTodo"></todo-detail>

Putting square bracket around the todo property, to the left of the equal to (=) sign, makes it the target of the property binding expression. You must declare a target binding property to be an input property. Otherwise, Angular rejects the binding and throws an error.

First amend the **@angular/core import** statement to include the Input statement in the todo-detail.component.ts file to include the **Input** symbol.

```
src/app/todo-detail.component.ts
--------------------------------------------------
import { Component, Input } from '@angular/core';
```

Then declare that todo is an *input* property by preceding it with the @Input decorator that you imported earlier.

```
src/app/todo-detail.component.ts
--------------------------------------------------
@Input() todo: Todo;
```

Ref : https://angular.io/docs/ts/latest/guide/attribute-directives.html#!#why-input

todo property is an input property

With this we are done, todo property is the only thing that is needed at this point in TodoDetailComponent class.

```
src/app/todo-detail.component.ts
--------------------------------------------------
export class TodoDetailComponent{
@Input() todo: Todo;
}
```

It receives the todo object through its todo input property and then bind to that property with its template.

todo property is an input property – match the code


```
src/app/todo-detail.component.ts
--------------------------------------------------
import { Component, Input } from '@angular/core';
import { Todo } from './todo';
@Component({
selector: 'todo-detail',
template:`
<div *ngIf="todo">
<h2>{{todo.name}} - details!</h2>
<div><label>id: </label>{{todo.id}}</div>
<div>
<label>name: </label>
<input [(ngModel)]="todo.name" placeholder="add task">
</div>
</div>
`
})
export class TodoDetailComponent{
@Input() todo: Todo;
}
```

Declare TodoDetailComponent in the AppModule

Every component must be declared in one—and only one—Angular module.

Open app.module.ts in your editor and import the TodoDetailComponent so you can refer to it.

```
src/app/app.module.ts
-------------------------------------------------
import { TodoDetailComponent } from './todo-detail.component';
```

Add TodoDetailComponent to the module's declarations array.

```
src/app/app.module.ts
-------------------------------------------------
declarations: [ AppComponent, TodoDetailComponent ],
```

In general, the declarations array contains a list of application components, pipes, and directives that belong to the module. A component must be declared in a module before other components can reference it. This module declares only the two application components, AppComponent and TodoDetailComponent.

Add TodoDetailComponent to AppComponent

The AppComponent is still a master/detail view. It used to display the todo details on its own, before you cut out that portion of the template. Now it will delegate to the TodoDetailComponent.

Recall that todo-detail is the CSS selector in the TodoDetailComponent metadata. That's the tag name of the element that represents the TodoDetailComponent.

Add a <todo-detail> element near the bottom of the AppComponent template, where the todo detail view used to be.

Coordinate the master AppComponent with the TodoDetailComponent by binding the selectedTodo property of the AppComponent to the todo property of the TodoDetailComponent.

src/app/app.component.ts

```
@Component({
selector: 'my-app',
template: `
<h1>{{title}}</h1>
<h2>My Todos</h2>
<ul class="todos">
<li *ngFor="let todo of todos" [class.selected]="todo === selectedTodo"
(click)="onSelect(todo)">
<span class="badge">{{todo.id}}</span>{{todo.name}}
</li>
</ul>
<todo-detail [todo]="selectedTodo"></todo-detail>
…….
```

Through this exercise what we have achieved?

As before, whenever a user clicks on a todo name, the todo detail appears below the todo list. But now the TodoDetailView is presenting those details.
Refactoring the original AppComponent into two components yields benefits, both now and in the future:

- You simplified the AppComponent by reducing its responsibilities.
- You can evolve the TodoDetailComponent into a rich todo editor without touching the parent AppComponent.
- You can evolve the AppComponent without touching the todo detail view.
- You can re-use the TodoDetailComponent in the template of some future parent component.

Thank You!

Email: info@yash.com
Web: www.yash.com