Tuesday: Creating Components

Creating Components

Till now, the *app* component has been handling everything that's in our application. We want to scale our app and we are therefore going to create more components. The beauty of components is that they enable us to decouple our application so that each functionality is implemented by a specific component. We end up having many components working together to achieve the whole purpose of the app.

Since it is the root components' job to render all other components, let us start out by creating a *goal* component to handle the logic on goals. In your terminal, run the following command:

```
$ ng generate component goal
```

This command has created a folder */goal* that has the files which contain the boilerplate code for the *goal* component. If we also check the *app.module.ts,* we notice that the goal component has been registered under `declarations` automatically by the angular CLI.

We're moving the goals logic from the *AppComponent* to the *GoalComponent*. Let's move the `goals` array permanently from the *AppComponent* class to the *GoalComponent* class*:*

src/app/goal/goal.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Goal } from '../goal';

@Component({
  selector: 'app-goal',
  templateUrl: './goal.component.html',
  styleUrls: ['./goal.component.css']
})
export class GoalComponent implements OnInit {

  goals:Goal[] = [
    {id:1, name:'Watch finding Nemo'},
    {id:2,name:'Buy Cookies'},
    {id:3,name:'Get new Phone Case'},
    {id:4,name:'Get Dog Food'},
    {id:5,name:'Solve math homework'},
    {id:6,name:'Plot my world domination plan'},
  ];

  constructor() { }

  ngOnInit() {
  }

}
```

At the top, we have imported the `Goal` blueprint class and then defined the array of goals inside the *GoalComponent* definition class. `ngOnInit` is a lifecycle hook. It is called each time the component is created. We use it to put complex initialization logic that we want for the component.

Let's also move the template logic to the *goal* Component.

src/app/goal/goal.component.html

```
<div>
  <h1>My Goals</h1>
  <hr>
  <ul>
    <li *ngFor='let goal of goals'>
      {{goal.name}}
    </li>
  </ul>
</div>
```

Nesting Components

Now that we have all our goals logic inside the *GoalComponent*, let's make it available as a child component of the root component, *AppComponent*. If we check our browser right now, there's nothing displaying because our root component has nothing to display. Using the *GoalComponent*'s selector, let's nest the *GoalComponent* inside the *AppComponent* so we can display the contents of the goal component.

src/app/app.component.html

```
<app-goal></app-goal>
```

If we check our browser now, the goals are displayed just like before. Yes, nesting is as simple as that! It ensures that the child component is loaded while inside the parent component.

Child Components

Let's add something more to our goals, let's give each goal a description. We will use the description as the detail of our goals and display it on our app. We will create a G*oalDetailComponent* which will be a child component of the *GoalComponent*. The G*oal-detailComponent* will be responsible for displaying the details of each goal which in our case will be the description of a goal.

Currently, our goal blueprint allows us to create goal objects with an `id` and `name` only. Let's add `description` to the goal blueprint so it allows us to create a description for each of our goals.

src/app/goal.ts

```
export class Goal {
  id: number;
  name: string;
  description: string;
}
```

We can now add descriptions to our goals in the array so that we can display them in our G*oalDetailComponent*.

src/app/goal/goal.component.ts

```
...
export class GoalComponent implements OnInit {
```

```
  goals: Goal[] = [
    {id:1, name:'Watch finding Nemo',description:'Find an online version and watch merlin find his
 son'},
    {id:2,name:'Buy Cookies',description:'I have to buy cookies for the parrot'},
    {id:3,name:'Get new Phone Case',description:'Diana has her birthday coming up soon'},
    {id:4,name:'Get Dog Food',description:'Pupper likes expensive sancks'},
    {id:5,name:'Solve math homework',description:'Damn Math'},
    {id:6,name:'Plot my world domination plan',description:'Cause I am an evil overlord'},
  ];

...
```

Input Property Binding

Our G*oalDetailComponent* will be receiving the goal description to display from the parent component, *GoalComponent*. For a child component to receive data from a parent component in Angular, we need to do input property binding. **Input property binding** allows us to pass data from a parent component to its child components. Let's see how we do this. On your terminal, use the CLI to generate a component named *goal-detail* and navigate to the G*oalDetailComponent* class file.

src/app/goal-detail/goal-detail.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { Goal } from '../goal';

@Component({
  selector: 'app-goal-detail',
  templateUrl: './goal-detail.component.html',
  styleUrls: ['./goal-detail.component.css']
})
export class GoalDetailComponent implements OnInit {

  @Input() goal: Goal;
  constructor() { }

  ngOnInit() {
  }

}
```

At the top, we add an import for `Input` from `@angular/core` which allows us to do Input property binding. We also import the `Goal` blueprint class. In the definition class for the G*oalDetailComponent*, we define `goal` as the property that will undergo input property binding, which is of the type `Goal`, from the blueprint class. This means when we will have the property `goal` in the G*oalDetailComponent*, it will have received its data from a parent component, in our case, the *GoalComponent*.

To bind to this `goal` property from the parent component *GoalComponent* to the child component *GoalDetailComponent* , we change our *GoalComponent* template code to this:

src/app/goal/goal.component.html

```
<div>
  <h1>My Goals</h1>
  <hr>
  <ul>
    <li *ngFor='let goal of goals'>
      {{goal.name}}
      <app-goal-detail [goal]='goal'></app-goal-detail>
    </li>
  </ul>
</div>
```

We have used the selector for the G*oalDetailComponent* and nested it into the G*oalComponent*, placing it after the goal name which is where we want to display the description of each goal. We have also added some logic in the child component tags and specified that we are binding the `goal` property to the G*oalDetailComponent*. The G*oalDetailComponent* is now receiving data from the G*oalComponent* so we now need to display this data in the G*oalDetailComponent*.

Summary of Input Binding

- The target in the square brackets, `[]`, is the property you decorate with `@Input()` in the child component. The binding source, the part to the right of the equal sign, is the data that the parent component passes to the nested component.
- The key takeaway is that when binding to a child component's property in a parent component—that is, what's in square brackets—you must decorate the property with `@Input()` in the child component.

src/app/goal-detail/goal-detail.component.html

```
<p>
  {{goal.description}}
</p>
```

When we serve our application at this point, we can see the application is now displaying each goal with its description below it.