As per Angular guidelines, it is a good coding practice to avoid too much complexity in a component. Hence, a single component should be given a single responsibility. This also means that our app will be made block-by-block using many components.

A parent component like *app.component* can have various child components embedded inside it.

What if we want to send some data which was selected by the user in the parent component, to a component that is inside the parent?

Component communication is needed if we want to share data between the components. Let us explore how to pass data from the container/parent component to the child component.

**Binding between the same component class and its template.**

We have already seen the components are actually custom HTML (template) tags.

For example, in the index.html, we used the AppComponent's template tag i.e, <app-root></app-root>.

* Inside the *app.component.html*, we can embed our child component called <app-book></app-book>.
* We could have had other default HTML components like <img> tags etc.

We have already seen property binding. For example, inside <app-root></app-root> (*app.component.html*), we have the following code.

1. <img [src]="iconUrl"/>
2. <button (click)="onSave()">Save</button>

The *iconUrl* and *onSave()* are members of the AppComponent class. We can always bind to a public property of a component in its own template.

Hence, a component's class and template are closely coupled. They are both parts of the same thing. Together they become the component.

What if we want to bind values of AppComponent (<app-root>) to a property of a child component, <app-book>

We would need to do something like this in app.component.html

1. <app-book [bookToDisplay]="userBook" ></app-book>

Compare this with the code given below (inside app.component.html)

1. <img [src]="iconUrl">

Here, both <img> and <app-book> are child components.

* Just like iconUrl is a property of AppComponent, userBook is also a property of AppComponent!
* Also just like, src is a property of the child <img> component, bookToDisplay is a property of the child <app-book> component. The only difference is that <app-book> is a custom component that we created!
* We are assigning the userBook stored in AppComponent to the bookToDisplay property of the child <app-book> component.

For the code <app-book [bookToDisplay]="userBook" ></app-book> to work, all we need is to tell Angular that the child component's **bookToDisplay** property will take value from its parent component.

This can be done using the ***@Input()*** annotation on the child component's property(ie, bookToDisplay). When the parent's userBook value is ready and assigned, the child will automatically get this value stored in the bookToDisplay.

# *****@Input*****

We can use ***@Input*** decorator in the child component on any property type like arrays, objects, etc.

The property of a component is considered as being private for binding with the property of other components. By writing **@Input()** decorator, we are making the property as public for binding from Angular perspective. After the use of the decorator, that property can receive input from any other components or directives.

So the input decorator should be used only when the binding is between 2 different component's properties.

**Example**: In the below example, we can observe how the username entered from the container component is passed to its child i.e. LoginComponent.

**app.component.html:**

Here in app.component.html the name entered by the user is being passed as a parameter to the login component at line 6.

1. <div style="text-align:center">
2. <h1>
3. Welcome to {{ title }}!
4. </h1>
5. <app-login [username]="firstname"></app-login>
6. </div>

**app.component.ts:**

At **Line 4**, AppComponent class contains an attribute to store the username entered in app.component.html.

1. ...
2. export class AppComponent {
3. title = 'app';
4. firstname= 'Jack';
5. }

**login.component.html :**

Line 6 contains the logic to display the username passed from AppComponent.

1. <p>
2. login works!
3. username is {{ username }}
4. </p>

**login.component.ts:**

At **Line 3**, **@Input** is used to receive the value from the container component and store it in **username**.

1. ...
2. export class LoginComponent {
3. @Input()
4. username : string;
5. }

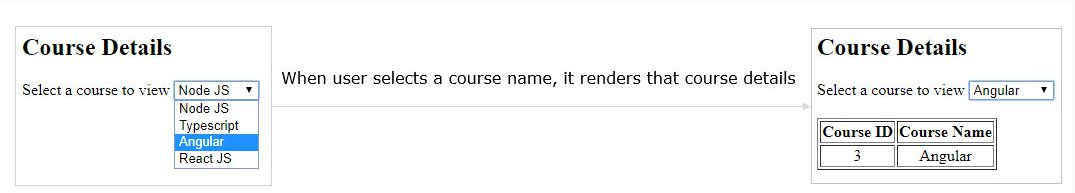
Now let us learn how to use @Input with the help of a demo.

Highlights:

* Loading nested component
* Passing data from a container to a child component

**Demo Steps:**

**Problem Statement:**Creating an AppComponent that displays a dropdown with a list of courses as values in it. Create another component called CoursesList component and load it in AppComponent which should display the course details. When the user selects a course from the dropdown,  corresponding course details should be loaded. The Output is as shown below:



1. Open **courses-list.component.ts** file created in nested components example and add the following code

1. import { Component, Input } from '@angular/core';
2. @Component({
3. selector: 'app-courses-list',
4. templateUrl: './courses-list.component.html',
5. styleUrls: ['./courses-list.component.css']
6. })
7. export class CoursesListComponent {
8. courses = [
9. { courseId: 1, courseName: "Node JS" },
10. { courseId: 2, courseName: "Typescript" },
11. { courseId: 3, courseName: "Angular" },
12. { courseId: 4, courseName: "React JS" }
13. ];
14. @Input() cName:string='';
15. }

**Line 18:** @Input() specifies that cName property will receive value from its container component.

We are making the property as public for binding from Angular perspective. After the use of the decorator, that property can receive input from any other components or directives.

1. Open **courses-list.component.html** and add the following code
2. <table border="1" \*ngIf="cName">
3. <thead>
4. <tr>
5. <th>Course ID</th>
6. <th>Course Name</th>
7. </tr>
8. </thead>
9. <tbody>
10. <tr \*ngFor="let c of courses">
11. <td \*ngIf="c.courseName == cName">{{c.courseId}}</td>
12. <td \*ngIf="c.courseName == cName">{{c.courseName}}</td>
13. </tr>
14. </tbody>
15. </table>

**Line 9-12:** ngFor iterates on courses array and displays courseId and courseName properties in a table

1. Add the following in **app.component.html**
2. <h2> Course Details </h2>
3. Select a course to view <select #course (change)="name = course.value">
4. <option value="Node JS">Node JS</option>
5. <option value="Typescript">Typescript</option>
6. <option value="Angular">Angular</option>
7. <option value="React JS">React JS</option>
8. </select><br/><br/>
9. <app-courses-list [cName]="name"></app-courses-list>

**Line 3-8:** It displays a dropdown to select a course to display its details. When the user selects a value, it assigns selected value to name property.

**Line 10:** This will load CoursesListComponent and passes the name property value to cName property of CoursesListComponent class.

**In app.componen.ts** initialize name to a string

name:string=" ";

     4. Run the application and check the output in the browser

Next, we will see how to pass data from a child component to a container component.

Now let us explore how to pass data from child to container component.

If a child component wants to send data to its parent component, then it must create a property with @Output decorator.

The only method for the child component to pass data to its parent component is through events. The property must be of type **EventEmitter**

**Example**: 2

In the below example, we can observe how the data is transferred from LoginComponent to AppComponent.

**login.component.ts:**

**Line 3,4 -**A custom event called **"customevent"** is created.

**Line 6-8 -**emitevent() is a method that emits a value 11 on change event.

1. ...
2. export class LoginComponent {
3. @Output()
4. customevent: EventEmitter<number> = new EventEmitter<number>();
5. emitevent(){
6. this.customevent.emit(11);
7. }
8. }

**login.component.html:**

Line 1, emitevent() is invoked on the click of a button.

1. <button (click)="emitevent()">Event emitter</button>

**app.component.ts:**

At line 5 updateFromChild() is used to receive the data from the child.

1. ...
2. export class AppComponent {
3. count: number = 0;
4. updateFromChild($event:number){
5. this.count = $event;
6. }
7. }

**app.component.html:**

Line 4, displays the data retrieved from the child component.

1. <div style="text-align:center">
2. <app-login (customevent)="updateFromChild($event)" ></app-login>
3. {{ count }}
4. </div>

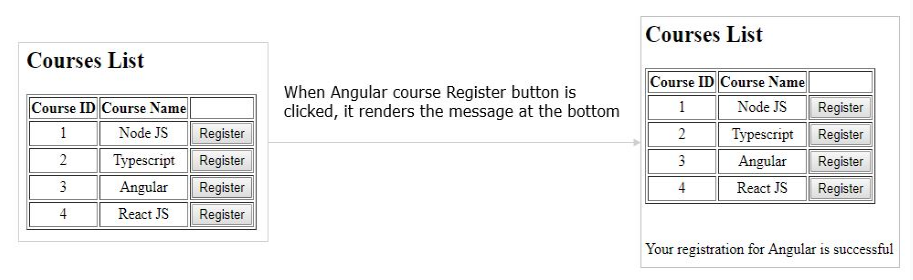
Let us now take a look at its implementation with the help of a demo.

Highlights:

* Loading nested component
* Passing data from child to container component

**Demo Steps:**

**Problem Statement:** Let us create an AppComponent that loads another component called CoursesList component. Create another component called CoursesListComponent which should display the courses list in a table along with a register button in each row. When a user clicks on the register button, it should send that courseName value back to AppComponent where it should display registration successful message along with courseName



1. Open the **courses-list.component.ts** file created in the previous example and add the following code

1. import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
2. @Component({
3. selector: 'app-courses-list',
4. templateUrl: './courses-list.component.html',
5. styleUrls: ['./courses-list.component.css']
6. })
7. export class CoursesListComponent {
8. @Output() onRegister = new EventEmitter<string>();
9. courses = [
10. { courseId: 1, courseName: "Node JS" },
11. { courseId: 2, courseName: "Typescript" },
12. { courseId: 3, courseName: "Angular" },
13. { courseId: 4, courseName: "React JS" }
14. ];
15. register(courseName: string) {
16. this.onRegister.emit(courseName);
17. }
18. }

**Line 9:** Create a property **onRegister**of type **EventEmitter**and attach **@Output** decorator which makes the property to send the data from child to parent

**Line 18:** This line emits the **courseName**value i.e. sends the courseName value back to parent component.

1. Open **courses-list.component.html** and add the following code
2. <table border="1">
3. <thead>
4. <tr>
5. <th>Course ID</th>
6. <th>Course Name</th>
7. <th></th>
8. </tr>
9. </thead>
10. <tbody>
11. <tr \*ngFor="let course of courses">
12. <td>{{course.courseId}}</td>
13. <td>{{course.courseName}}</td>
14. <td><button (click)="register(course.courseName)">Register</button></td>
15. </tr>
16. </tbody>
17. </table>

**Line 13:** When the user clicks this button, it invokes the register method by passing the courseName value.

1. Add the following in **app.component.html**
2. <h2> Courses List </h2>
3. <app-courses-list (onRegister)="courseReg($event)"></app-courses-list>
4. <br/><br/>
5. <div \*ngIf="message">{{message}}</div>

**Line 3:** Binds onRegister event with courseReg method of the parent component. When CoursesListComponent emits the value, onRegister event is triggered and it invokes courseReg method. $event holds the value emitted by CoursesListComponent

**Line 6:** This renders the message property value which holds the value emitted

1. Add the following code in **app.component.ts**
2. import { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9. message: string="";
10. courseReg(courseName: string) {
11. this.message = `Your registration for ${courseName} is successful`;
12. }
13. }

**Line 12:** courseReg method is invoked when onRegister event emits

**Line 13:** Assigns the string to message property which will be rendered in the template

5. Run the application and check the output in the browser