## **Steps to create basic Component**

1 Check the angular version

Ng version

**D:\angComponentExample>ng new baseapp --no-standalone --routing --ssr=false**

## Cd baseapp

Ng g c testcmp

1. Edit testcmp.component.css

Add below code

h1{

background-color: aqua;

color: purple;

font-size: 25px;

}

## 2.edit testcmp.component.ts and add below code

import { Component } from '@angular/core';

@Component({

selector: 'app-testcmp',

templateUrl: './testcmp.component.html',

styleUrl: './testcmp.component.css'

})

export class TestcmpComponent {

name = "Swami Narayan";

}

3. Edit testcmpcomponent.html and add below code

<p>Hello inside .... testcmp works!{{name}}</p>

## 4. Add to app.module.ts file below code

## **import { NgModule } from '@angular/core';**

## **import { BrowserModule } from '@angular/platform-browser';**

## 

## **import { AppRoutingModule } from './app-routing.module';**

## **import { AppComponent } from './app.component';**

## **import { TestcmpComponent } from './testcmp/testcmp.component';**

## 

## **@NgModule({**

## **declarations: [**

## **AppComponent,**

## **TestcmpComponent**

## **],**

## **imports: [**

## **BrowserModule,**

## **AppRoutingModule**

## **],**

## **providers: [],**

## **bootstrap: [AppComponent]**

## **})**

## **export class AppModule { }**

## 

## 

5. Add below code to app.component.html

<app-testcmp></app-testcmp>

## 

6. Ng serve

## 

## **Attribute Directives in Angular example**

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

template: `

<p [ngClass] ="['warning','big']">

<ng-content></ng-content></p>`,

/\* <p ngClass="centered-text underlined" class= "orange">

<ng-content></ng-content></p> \*/

styles: [`

.warning{

color:red;

font-weight:bold;

}

.big{

font-size:1.2rem;

}

/\* .centered-text {

text-align:center;

}

.underlined{

border-bottom:1px solid #ccc;

}

.orange{

color:orange;

} \*/

`]

})

export class AppComponent {

//title = 'ngclassdirectiveapp';

}

## **import { Component } from '@angular/core';**

## 

## **@Component({**

## **selector: 'app-root',**

## **template: `**

## **<ul><li \*ngFor= "let item of courses; let i = index">**

## **<p> {{i+1}} - {{item.name}}</p></li> </ul>`})**

## **/\* <button type="button" (click)="toggleExists()">**

## **Toggle Component</button>**

## **<hr>**

## **<app-root \*ngIf="exists">**

**Structural Directive Example of NgFor and NgIf**

## **Welcome to ngIf**

## **</app-root> `}) \*/**

## **export class AppComponent {**

## **// title = 'ngIfdirectiveapp';**

## 

## **/\* exists =true;**

## **toggleExists(){**

## **this.exists = !this.exists;**

## **} \*/**

## **courses : any[]=[**

## **{"name": "Java"},{"name":"Python"},{"name": "CSharp"},{"name":"Angular"} ];**

## **}**

## 

## 

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

template: `

<h4> NgIf and NgFor Example </h4>

<ul \*ngFor= "let user of Users">

<li \*ngIf = "user.age < 50">

<p> {{user.name}} age is ({{user.age}}) </p>

</li>

</ul>

`

})

export class AppComponent {

Users : any[]=[{"name":"Ravikant","age": 23,"city":"Chennai" }

,{"name":"Hari Krishna","age":35,"city":"Vadtal"},

{"name":"Chintan Swami","age":48,"city":"Surendra Nagar"},

{"name":"Laxmi","age":34,"city":"Mumbai"}];

}

## **NgSwitch**

import { Component } from '@angular/core';

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

@Component({

selector: 'app-root',

template: `

<h4> NgSwitch </h4>

<ul \*ngFor= "let user of Users"

[ngSwitch] = "user.city">

<li \*ngSwitchCase = "'Vadtal'">

<p>Happy</p>

<p> {{user.name}} age is ({{user.age}}) </p>

</li>

<li \*ngSwitchCase = "'Mumbai'"

class="text-primary">

<p> {{user.name}} age is ({{user.age}}) </p>

</li>

<li \*ngSwitchCase = "'Surendra Nagar'"

class="text-Primary">

<p> {{user.name}} age is ({{user.age}}) </p>

</li>

</ul>

`

})

export class AppComponent {

Users : any[]=[{"name":"Ravikant","age": 23,"city":"Chennai" }

,{"name":"Hari Krishna","age":35,"city":"Vadtal"},

{"name":"Chintan Swami","age":48,"city":"Surendra Nagar"},

{"name":"Laxmi","age":34,"city":"Mumbai"}];

}

## **BuiltIn Pipes**

import { Component } from '@angular/core';

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

@Component({

selector: 'app-root',

template: `<div>

<p ngNonBindable> {{1167.34 | currency : 'USD' }}</p>

<p>{{1167.34 | currency : 'USD' }}</p>

<p ngNonBindable> {{1167.34 | currency : 'USD': 'symbol' }}</p>

<p>{{1167.34 | currency : 'USD' : 'symbol' }}</p>

<p ngNonBindable> {{1167.34 | currency : 'USD': 'symbol-narrow' }}</p>

<p>{{1167.34 | currency : 'USD' : 'symbol-narrow' }}</p>

<p ngNonBindable> {{dateVal | date: 'shortTime'}}</p>

<p> {{dateVal | date: 'shortTime'}} </p>

<p ngNonBindable> {{dateVal | date: 'fullDate'}}</p>

<p> {{dateVal | date: 'fullDate'}} </p>

<p ngNonBindable> {{dateVal | date: 'd/M/y'}}</p>

<p> {{dateVal | date: 'd/M/y'}} </p>

<p ngNonBindable> {{5.678890 | number: '3.1-3'}}</p>

<p> {{5.678890| number: '3.1-3'}} </p>

<p ngNonBindable> {{'anGuLAR piPE' | uppercase }}</p>

<p> {{'anGuLAR piPE' | uppercase}} </p>

</div>

`

})

export class AppComponent {

dateVal: Date =new Date();

}

Custom Pipe

## 

## **What is an Angular Component**

# The Angular Component is the main building block of an Angular application.

# The Component contains the data and user interaction logic that define how the View looks and behaves. A view in Angular refers to a template (HTML).

# The Angular Components are plain [JavaScript](https://www.tektutorialshub.com/javascript-tutorial/) classes defined using the **@Component Decorator**. This Decorator provides the Component with a View to display and Metadata about the Component.

# The Component is responsible for providing the data to the View. Angular uses [data binding](https://www.tektutorialshub.com/angular/angular-data-binding/) to get the data from the Component to the View and vice versa. The data binding is achieved using special HTML markup, the **Angular Template Syntax**. The Component can also get notified when the View changes.

# Angular applications can have lots of components. Each Component handles a small part of the UI. These components work together to produce the complete user interface of the application.

# The Components consist of three main building blocks

# Template

# Class

# MetaData

# Building Blocks of Angular Component Template, Metadata and Class

## **Building blocks of the Angular Components**

### **Template (View)**

# The Template defines the layout and content of the View. Without the template, there is nothing for Angular to render to the DOM.

# The Templates are only HTML codes and the Angular-specific special HTML markups (known as the Angular Template Syntax).

# You can add [Angular directives](https://www.tektutorialshub.com/angular/angular-directives/), [Angular Pipes](https://www.tektutorialshub.com/angular/angular-pipes/) & Other Angular Components on the template.

# The data to Template comes from the Component, which gets it from an [Angular Service](https://www.tektutorialshub.com/angular/angular-services/). We can keep the Template in sync with the Component using the data binding techniques. The templates can use [Event Binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) or [two-way data binding](https://www.tektutorialshub.com/angular/ngmodel-two-way-data-binding-in-angular/) to notify the component when the user changes something on the View.

# There are two ways you can specify the Template in Angular.

# Defining the Template Inline

# Provide an external Template

### **Class**

# The Class provides the data & logic to the View. It contains the [Typescript](https://www.tektutorialshub.com/typescript-tutorial/) code associated with Template (View). We use [TypeScript](https://www.tektutorialshub.com/typescript-tutorial/) to create the class.

# Class Contains the Properties & Methods. The Properties of a class can be bound to the view using [Data Binding](https://www.tektutorialshub.com/angular/angular-data-binding/).

# The simple Angular Class

| 123456 | export **class** AppComponent{ title : **string** ="app"} |
| --- | --- |

# By convention, we suffix the Component class as Component to easily identify them.

### **Metadata**

# Metadata Provides additional information about the component of Angular. Angular uses this information to process the class. We use the @Component decorator to provide the Metadata to the Component.

#### **@Component decorator**

# A decorator is a function that adds metadata to a class, its methods & to its properties. The Components are defined with a @Component class decorator.

# When Angular sees a class with @Component decorator, it treats the class as a Component.

# A Decorator is always prefixed with @. We must place the Decorator immediately before the class definition. We can also build our own decorators. The decorators are Similar to attributes in C#

#### **Important Component metadata properties**

##### **Selector**

# Selector specifies the simple CSS selector. The Angular looks for the CSS selector in the template and renders the component there.

##### **Providers**

# The Providers are the [Angular Services](https://www.tektutorialshub.com/angular/angular-services/) that our component is going to use. The Services provide service to the Components or to the other Services.

##### **Directives**

# The [directives](https://www.tektutorialshub.com/angular/angular-directives/) that this component going to use are listed here.

##### **Styles/styleUrls**

# The CSS styles or style sheets that this component needs. Here we can use either an external stylesheet (using styleUrls) or inline styles (using Styles). The styles used here are specific to the component

##### **template/templateUrl**

# The HTML template that defines our View. It tells Angular how to render the Component’s view. The templates can be inline (using a template) or we can use an external template (using a templateUrl). The Component can have only one template. You can either use an inline template or an external template, and not both.

## **How to Create Angular Component**

# Create a new Angular application. If you are new to Angular, you can follow the steps provided in the following tutorial to create the application

# [Install Angular](https://www.tektutorialshub.com/angular/angular-installing-settingup-development-environment/)

# [Create Angular Project](https://www.tektutorialshub.com/angular/angular-create-first-application/)

# When we create a new project, Angular automatically creates the component **app.component.ts**. It will also mark this as a root component, so when you run the project, it will display its template (**app.component.html**) to the user.

# In this tutorial, we add another angular component and make it the root component. We name our Component **HelloWorldComponet**. When we run the App, it will display contents from the newly created Component.

# Creating an Angular Component requires us to follow these steps

# Create the Component file.

# Import the required external Classes/Functions.

# Create the Component class and export it.

# Add @Component decorator

# Add metadata to @Component decorator

# Create the Template

# Create the CSS Styles

# Register the Component in Angular Module

### **Create the Component file**

# Select the **src/app** folder, **right-click**, and **click on the new file**. Enter the name of the file as **hello-world.component.ts**

# The file name starts with the **feature name** (hello-world) by convention. We then follow it up with a dot and **type of class** (Component). The extension used is **.ts** indicating that this is a typescript module file.

# You can read more about naming conventions from the [Angular Style Guide](https://angular.io/guide/styleguide).

### **Import the Angular Component Library**

# Before we use any Angular (or external) functions or classes, we need to tell Angular how and where to find them. We do this using the Import statement. The Import statement is similar to the using statement in c#. It allows us to use the external modules in our class.

# To define the Component class, we need to use the @Component decorator. This function is part of the Angular Core library. So we import it into our class as shown below

# import { Component } from '@angular/core';

### **Create the Component Class and export it**

# The third step is to create the Component class using the export keyword. The Export keyword allows other components to use this Component by importing it.

| 12345 | export **class** HelloWorldComponent { title = 'Hello World';} |
| --- | --- |

# Note we are using Pascal case naming conventions for the class name. The above class defines a Property named title and assigns a default value, “Hello World”.

### **Add @Component decorator**

# The next step is to inform Angular that this is a Component class. We do that by adding the @Component decorator. We must add the decorator immediately above the class definition.

| 1234567 | @Component({})export **class** HelloWorldComponent { title = 'Hello World';} |
| --- | --- |

### **Add metadata to @Component decorator**

# The next step is to add the metadata to the Component using the @component decorator. Add the following to the component metadata.

| 12345678910 | @Component({ selector: 'app-hello-world', templateUrl: './hello-world.component.html', styleUrls: ['./hello-world.component.css']})export **class** HelloWorldComponent { title = 'Hello World';} |
| --- | --- |

#### **selector**

# We have chosen app-hello-world as the selected for this Component. The Angular, wherever it finds the selector app-hello-world in the HTML, renders the HelloWorldComponent in its place.

#### **templateUrl**

# In the above example, we have used an external template using templateUrl metadata—the templateUrl points to the external HTML file **hello-world.component.html**.

# We can specify only one file as a template file per Component.

#### **styleUrls**

# styleUrls declares the styles for our Template. The metadata points to the external CSS file **hello-world.component.css**. This file is a Component specific CSS style.

# We can include more than one CSS file.

### **Create the Template (View)**

# The Template is an HTML file, which the Component must display to the user.

# The Angular knows which template display, using the **templateUrl** metadata, which points to **hello-world.component.html**.

# Select the **src/app** folder, **right-click**, and **click on the new file**. Enter the name of the file as **hello-world.component.html.**

# By convention, we give the same name as the component file it represents. It will have an extension of HTML.

| 12345 | <h1> {{title}}!</h1> |
| --- | --- |

# Note that **title***is*inside the **double curly bracket**. When rendering the View, Angular looks for the **title Property** in our Component and binds the property to our View. We call this [data binding](https://www.tektutorialshub.com/angular/angular-data-binding/). The double curly bracket syntax is known as [interpolation](https://www.tektutorialshub.com/angular/angular-data-binding/#Interpolation), which we will look at in our following tutorial.

### **Add the Styles**

# The next step is to add the CSS Styles. The styleUrls metadata tells Angular where to find the CSS File. This property points to external file **hello-world.component.css**

# By convention, we name the file after the component file with the .css extension. Note that styleUrls metadata can accept multiple CSS Files.

# Select the **src/app** folder, **right-click**, and **click on the new file**. Enter the name of the file as **hello-world.component.css**

# Let us keep the file blank for now.

### **Register the Component in Angular Module**

# We have created the Angular Component. The next step is to register it with the [Angular Module](https://www.tektutorialshub.com/angular/angular-modules/). Our application already has one Module, i.e., app.module.ts, which is also a root module.

# The root module is the Module Angular loads when the App starts.

# The [Angular Module](https://www.tektutorialshub.com/angular/angular-modules/) organizes the related components, directives, pipes, and services and arranges them into cohesive blocks of functionality.

# We use the [@NgModule](https://www.tektutorialshub.com/angular/angular-modules/) class decorator to define an [Angular Module](https://www.tektutorialshub.com/angular/angular-modules/) and provide metadata about the Modules. There are four important properties of @NgModule metadata. They are **declaration**, **imports**, **providers** & **bootstrap.**

# We include the components, [pipes](https://www.tektutorialshub.com/angular/angular-pipes/), and [directives](https://www.tektutorialshub.com/angular/angular-directives/) part of this Module in the **declaration array**.

# We add all the other [Angular Modules](https://www.tektutorialshub.com/angular/angular-modules/) this Module needs in the **imports array**.

# Include all the [angular services](https://www.tektutorialshub.com/angular/angular-services/) part of this Module in the [**providers’**](https://www.tektutorialshub.com/angular/angular-providers/) **array**.

# Assign the Component we want to load when Angular loads this Module to the **bootstrap array**.

# To Make use of our HelloWorldComponent, we import it.

| 123 | import { HelloWorldComponent } from './hello-world.component'; |
| --- | --- |

# We also need to add it to the **declaration array**.

| 12345 | declarations: [ AppComponent, HelloWorldComponent ], |
| --- | --- |

# Now, our Component is ready to use.

# Our final app.module.ts.

| 123456789101112131415161718192021 | import { NgModule } from '@angular/core';import { BrowserModule } from '@angular/platform-browser';import { AppRoutingModule } from './app-routing.module';import { AppComponent } from './app.component';import { HelloWorldComponent } from './hello-world.component';@NgModule({ declarations: [ AppComponent, HelloWorldComponent ], imports: [ BrowserModule, AppRoutingModule ], providers: [], bootstrap: [AppComponent]})export **class** AppModule { } |
| --- | --- |

# How to Create Angular Component

## **HelloWorldComponent as Root Component**

# When the application starts, It loads the AppComponent. Let us change it and display HelloWorldComponent instead.

# To do that, we assign the HelloWorldComponent Component to the bootstrap property and remove the AppComponent.

# You can just open the app.module.ts and locate the following code.

| 123 | bootstrap: [AppComponent] |
| --- | --- |

# Update it with the following code.

| 123 | bootstrap: [HelloWorldComponent] |
| --- | --- |

# Now open the **src/app/index.html** and locate the following line.

| 12345 | <body><app-root></app-root></body> |
| --- | --- |

# and replace it with this

| 12345 | <body><app-hello-world></app-hello-world></body> |
| --- | --- |

# Run the App using ng serve --open. You will see **“Hello World**” on the screen.

# HelloWorldComponent as Root Angular Component

## **Multiple Root Components**

# Angular allows us to bootstrap multiple components. We can make both HelloWorldComponent and AppComponent as root components.

# We need to update the bootstrap array property of the app.module.ts and also add both selectors in **index.html**.

# Update the **bootstrap property** in the app.module.ts.

| 123 | bootstrap: [AppComponent, HelloWorldComponent] |
| --- | --- |

# Add both selectors in the **index.html**

| 123456 | <body> <app-hello-world></app-hello-world> <app-root></app-root></body> |
| --- | --- |

# Open the **app.component.html** and update the content to

| 123 | <h1>Hello **for** AppComponent</h1> |
| --- | --- |

# Run the application.

# Multiple Root Angular Components in Angular

## **Display as a Child Component**

# We rendered the HelloWorldComponet as the root component. But this is not the correct way to render a child component. You should have only one Component as the root Component. All other Components must be rendered as the child of the root component.

# There are two ways in which we can display a [child Component](https://www.tektutorialshub.com/angular/angular-adding-child-component/).

# Via [Angular Router](https://www.tektutorialshub.com/angular/angular-routing-navigation/)

# Via Component Selector

# The [Angular Router Tutorial](https://www.tektutorialshub.com/angular/angular-routing-navigation/) covers rendering a Component using the Angular Router.

# We already used the component selector to render the Component ( app-hello-world for HelloWorldComponent). To render HelloWorldComponent as child component follow the following steps

# Remove the <app-hello-world></app-hello-world> from the **index.html**.

# Remove the HelloWorldComponent from the **bootstrap property** in app.module.ts.

# Open the app.component.html and add the app-hello-world selector.

| 1234 | <h1>Hello **for** AppComponent</h1><app-hello-world></app-hello-world> |
| --- | --- |

# Now the HelloWorldComponent renders as a child component of the AppComponent.

## **Summary**

# Follow these steps to create the Angular Component

# Create the Component file.

# Import the required external Classes/Functions.

# Create the Component class and export it.

# Add @Component decorator

# Add metadata to @Component decorator

# Create the Template

# Create the CSS Styles

# Register the Component in Angular Module

# There are two ways in which you can display the Component. One is using the Angular Router, and another way is to use the Component selector in the Template.

# 

# 

# Angular components overview

# Components are the main building blocks for Angular applications. Each component consists of:

# An HTML template that declares what renders on the page

# A TypeScript class that defines behaviour

# A CSS selector that defines how the component is used in a template

# Optionally, CSS styles applied to the template

# This topic describes how to create and configure an Angular component.

## Prerequisites

# To create a component, verify that you have met the following prerequisites:

# [Install the Angular CLI.](https://angular.io/guide/setup-local#install-the-angular-cli)

# [Create an Angular workspace](https://angular.io/guide/setup-local#create-a-workspace-and-initial-application) with initial application. If you don't have a project, create one using ng new <project-name>, where <project-name> is the name of your Angular application.

## Creating a component

# The best way to create a component is with the Angular CLI. You can also create a component manually.

### Creating a component using the Angular CLI

# To create a component using the Angular CLI:

# From a terminal window, navigate to the directory containing your application.

# Run the ng generate component <component-name> command, where <component-name> is the name of your new component.

# By default, this command creates the following:

# A directory named after the component

# A component file, <component-name>.component.ts

# A template file, <component-name>.component.html

# A CSS file, <component-name>.component.css

# A testing specification file, <component-name>.component.spec.ts

# Where <component-name> is the name of your component.

# You can change how ng generate component creates new components. For more information, see [ng generate component](https://angular.io/cli/generate#component-command) in the Angular CLI documentation.

### Creating a component manually

# Although the Angular CLI is the best way to create an Angular component, you can also create a component manually. This section describes how to create the core component file within an existing Angular project.

# To create a new component manually:

# Navigate to your Angular project directory.

# Create a new file, <component-name>.component.ts.

# At the top of the file, add the following import statement.

# content\_copy

# import { [Component](https://angular.io/api/core/Component) } from '@angular/core';

# After the import statement, add a @[Component](https://angular.io/api/core/Component) decorator.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# })

# Choose a CSS selector for the component.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# })

# For more information on choosing a selector, see [Specifying a component's selector](https://angular.io/guide/component-overview#specifying-a-components-css-selector).

# Define the HTML template that the component uses to display information. In most cases, this template is a separate HTML file.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# templateUrl: './component-overview.component.html',

# })

# For more information on defining a component's template, see [Defining a component's template](https://angular.io/guide/component-overview#defining-a-components-template).

# Select the styles for the component's template. In most cases, you define the styles for your component's template in a separate file.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# templateUrl: './component-overview.component.html',

# styleUrls: ['./component-overview.component.css']

# })

# Add a class statement that includes the code for the component.

# content\_copy

# export class ComponentOverviewComponent {

# 

# }

## Specifying a component's CSS selector

# Every component requires a CSS *selector*. A selector instructs Angular to instantiate this component wherever it finds the corresponding tag in template HTML. For example, consider a component hello-world.component.ts that defines its selector as app-hello-world. This selector instructs Angular to instantiate this component any time the tag <app-hello-world> appears in a template.

# Specify a component's selector by adding a selector property to the @[Component](https://angular.io/api/core/Component) decorator.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# })

## Defining a component's template

# A template is a block of HTML that tells Angular how to render the component in your application. Define a template for your component in one of two ways: by referencing an external file, or directly within the component.

# To define a template as an external file, add a templateUrl property to the @[Component](https://angular.io/api/core/Component) decorator.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# templateUrl: './component-overview.component.html',

# })

# To define a template within the component, add a template property to the @[Component](https://angular.io/api/core/Component) decorator that contains the HTML you want to use.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# template: '<h1>Hello World!</h1>',

# })

# If you want your template to span multiple lines, use backticks (`). For example:

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# template: `

# <h1>Hello World!</h1>

# <p>This template definition spans [multiple](https://angular.io/api/forms/SelectMultipleControlValueAccessor) lines.</p>

# `

# })

# An Angular component requires a template defined using template or templateUrl. You cannot have both properties in a component.

## Declaring a component's styles

# Declare component styles used for its template in one of two ways: By referencing an external file, or directly within the component.

# To declare the styles for a component in a separate file, add a styleUrls property to the @[Component](https://angular.io/api/core/Component) decorator.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# templateUrl: './component-overview.component.html',

# styleUrls: ['./component-overview.component.css']

# })

# To declare the styles within the component, add a styles property to the @[Component](https://angular.io/api/core/Component) decorator that contains the styles you want to use.

# content\_copy

# @[Component](https://angular.io/api/core/Component)({

# selector: 'app-component-overview',

# template: '<h1>Hello World!</h1>',

# styles: ['h1 { font-weight: normal; }']

# })

# The styles property takes an array of strings that contain the CSS rule declarations.

# 

# 

# 

# 

# Angular Component Output Properties

## **Angular Component Output Properties**

In this article, I am going to discuss **Angular Component Output Properties** in detail. At the end of this article, you will have a very good understanding of the below things.

1. **How to pass the user actions or user entered values or selections from the child component to the parent component using output properties.**
2. **How to create custom events using angular EventEmitter class**
3. **What are ng-container directive and its use**

##### **What are Angular Component Output Properties?**

The **Angular Component Output Properties** are used to to pass the data from the nested component to the container component.

This is a continuation part of our [**previous article**](https://dotnettutorials.net/lesson/angular-component-input-properties/), so please read our previous article before proceeding to this article as we are going to work with the same example as shown below.

Angular Component Output Properties

At the moment when you click any of the radio buttons, then nothing is happening. Let first discuss what we want to do here.

When All(6) radio button is clicked then we need to display all the students in the table. When Male(4) radio button is clicked then we need to display only the 4 Male students in the table similarly when the Female(2) radio button is clicked then we only need to display the 2 Female students in the table.

##### **How to achieve this?**

To achieve this here we are going to make use of the **Angular component Output Properties**. First, let’s discuss the changes that are required in the nested component i.e. **StudentCountComponent**.

##### **Modify student-count.component.ts file:**

Open student-count.component.ts file and then copy and paste the following code in it. The changes we have done here are self-explained so please go through the comments.

// Import Output and EventEmitter from angular

**import** **{** Component, Input, Output, EventEmitter **}** from '@angular/core';

@Component**({**

selector: 'app-student-count',

templateUrl: './student-count.component.html',

styleUrls: **[**'./student-count.component.css'**]**

**})**

**export** **class** StudentCountComponent **{**

@Input**()**

all: number;

@Input**()**

male: number;

@Input**()**

female: number;

// This variable holds the selected value of the radio button

selectedRadioButtonValue: string = 'All';

// The Output decorator makes the property of an Component as an Output property

// The EventEmitter class in Angular is used to create the custom event

// When the radio button selection changes, the selected radio button

// value which is a string gets passed to the event handler method.

// Hence, the event payload is string.

@Output**()**

countRadioButtonSelectionChanged: EventEmitter<string> =

new EventEmitter<string>**()**;

// This method raises the custom event. We will bind this

// method to the change event of all the 3 radio buttons

onRadioButtonSelectionChange**()** **{**

this.countRadioButtonSelectionChanged

.emit**(**this.selectedRadioButtonValue**)**;

**}**

**}**

##### **Modify student-count.component.html**

The following are the changes that are required in the view template of StudentCountComponent i.e. **student-count.component.html**. Notice here we have made 3 changes on each radio button.

1. The value attribute is set to (All, Male or Female)
2. We implemented the 2-way data-binding using the ngModel directive. Notice the ngModel is bound to selectedRadioButtonValue property of the component class. This 2-way data-binding ensures whenever the radio button selection changes, the selectedRadioButtonValue property is updated with the value of the selected radio button.
3. The onRadioButtonSelectionChange() method is binded to the “change” event of the radio button. That means whenever, the selection of the radio button changes, onRadioButtonSelectionChange() method raises the custom event “countRadioButtonSelectionChanged”. We defined this custom event using Angular EventEmitter class.

**<span** class="radioClass"**>**Show : **</span>**

**<input** name='options' type='radio' value="All"

[(ngModel)]="selectedRadioButtonValue"

(change)="onRadioButtonSelectionChange()"**>**

**<span** class="radioClass"**>**{{'All(' + all + ')'}}**</span>**

**<input** name="options" type="radio" value="Male"

[(ngModel)]="selectedRadioButtonValue"

(change)="onRadioButtonSelectionChange()"**>**

**<span** class="radioClass"**>**{{"Male(" + male + ")"}}**</span>**

**<input** name="options" type="radio" value="Female"

[(ngModel)]="selectedRadioButtonValue"

(change)="onRadioButtonSelectionChange()"**>**

**<span** class="radioClass"**>**{{"Female(" + female + ")"}}**</span>**

##### **Modify student-list.component.ts file**

Now let’s have a look at the changes that are required in the parent component i.e. StudentListComponent. The following are the changes that are required in the StudentListComponent class. The changes are commented and self-explained.

**import** **{** Component**}** from '@angular/core';

@Component**({**

selector: 'app-student-list',

templateUrl: './student-list.component.html',

styleUrls: **[**'./student-list.component.css'**]**

**})**

**export** **class** StudentListComponent **{**

// This property will keep track of the radio button which is selected

// We have set the default value to All, so all the students

// are displayed in the table by default

selectedStudentCountRadioButton: string = 'All';

students: any**[]** = **[**

**{**

ID: 'std101', FirstName: 'Pranaya', LastName: 'Rout',

DOB: '12/8/1988', Gender: 'Male', CourseFee: 1234.56

**}**,

**{**

ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty',

DOB: '10/14/1989', Gender: 'Male', CourseFee: 6666.00

**}**,

**{**

ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',

DOB: '7/24/1992', Gender: 'Female', CourseFee: 6543.15

**}**,

**{**

ID: 'std104', FirstName: 'Hina', LastName: 'Sharma',

DOB: '8/19/1990', Gender: 'Female', CourseFee: 9000.50

**}**,

**{**

ID: 'std105', FirstName: 'Sambit', LastName: 'Satapathy',

DOB: '4/12/1991', Gender: 'Male', CourseFee: 9876.54

**}**,

**{**

ID: 'std106', FirstName: 'Tarun', LastName: 'Mallick',

DOB: '4/10/1992', Gender: 'Male', CourseFee: 1278.55

**}**

**]**;

getTotalStudentsCount**()**: number **{**

**return** this.students.length;

**}**

getMaleStudentsCount**()**: number **{**

**return** this.students.filter**(**std => std.Gender === 'Male'**)**.length;

**}**

getFemaleStudentsCount**()**: number **{**

**return** this.students.filter**(**std => std.Gender === 'Female'**)**.length;

**}**

// Depending on the radio button which is selected, this method updates

// the selectedStudentCountRadioButton property

// This method is going to called when the child component (StudentCountComponent)

// raises the custom event - countRadioButtonSelectionChanged

// The event binding is specified in StudentList.component.html

onStudentCountRadioButtonChange**(**selectedRadioButtonValue: string**)**: void **{**

this.selectedStudentCountRadioButton = selectedRadioButtonValue;

**}**

**}**

##### **Modify student-list.component.html**

The following are the changes that are required in the view template of StudentListComponent i.e. student-list.component.html.

The **onStudentCountRadioButtonChange($event)** method is bound to the custom event – countRadioButtonSelectionChanged. The $event object will have the selected radio button value as that is what is passed as the event payload from the nested component. The event handler method (onStudentCountRadioButtonChange()) in the component class updates the property “selectedStudentCountRadioButton”. This property is then used along with the \*ngIf structural directive to decide which student objects to display in the table.

On the <tr> element, we are using “ngIf” directive along with selectedStudentCountRadioButton property which controls the student objects to display.

Notice, just above the <tr> element, we have introduced <ng-container> element and the “ngFor” directive is placed on this element. If you are wondering why we have done this, Angular does not allow multiple structural directives to be placed on one element as shown below.

**<app-student-count** [all]="getTotalStudentsCount()"

[male]="getMaleStudentsCount()"

[female]="getFemaleStudentsCount()"

(countRadioButtonSelectionChanged)="onStudentCountRadioButtonChange($event)"**>**

**</app-student-count>**

**<br/><br/>**

**<table>**

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**Name**</th>**

**<th>**Gender**</th>**

**<th>**DOB**</th>**

**<th>**Course Fee**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<ng-container** \*ngFor="let student of students;"**>**

**<tr** \*ngIf="selectedStudentCountRadioButton=='All' ||

selectedStudentCountRadioButton==student.Gender"**>**

**<td>**{{student.ID | uppercase}}**</td>**

**<td>**{{student.FirstName | uppercase}}**</td>**

**<td>**{{student.Gender}}**</td>**

**<td>**{{student.DOB | date:'dd/MM/y'}}**</td>**

**<td>**{{student.CourseFee | currency:'USD':true:'1.2-2'}}**</td>**

**</tr>**

**</ng-container>**

**<tr** \*ngIf="!students || students.length==0"**>**

**<td** colspan="10"**>**

No Students to display

**</td>**

**</tr>**

**</tbody>**

**</table>**

##### **Modify app.module.ts:**

Open app.module.ts file and then copy and paste the following code in it.

**import** **{** BrowserModule **}** from '@angular/platform-browser';

**import** **{** NgModule **}** from '@angular/core';

**import** **{** FormsModule **}** from '@angular/forms';

**import** **{** AppRoutingModule **}** from './app-routing.module';

**import** **{** AppComponent **}** from './app.component';

**import** **{** StudentListComponent **}** from './student-list/student-list.component';

**import** **{** StudentCountComponent **}** from './student-count/student-count.component';

@NgModule**({**

declarations: **[**

AppComponent,

StudentListComponent,

StudentCountComponent

**]**,

imports: **[**

BrowserModule,

AppRoutingModule,

FormsModule

**]**,

providers: **[]**,

bootstrap: **[**StudentListComponent**]**

**})**

**export** **class** AppModule **{** **}**

That’s it. We have done with our implementation. Now save all the changes and then run the application and you will see based on the selected radio button, the students are displayed in the table.

how to pass the data from the nested component to the container component using the Angular Component Output Properties