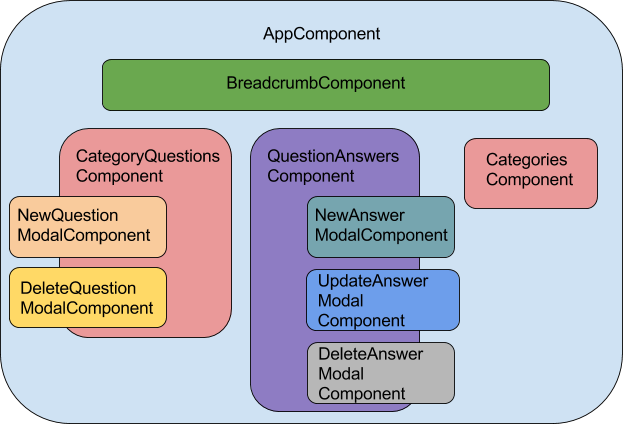
# Angular Components

* The component is the main building block of Angular applications.
* Components are composable, we can build larger components using multiple smaller components.
* Angular is basically a component-based MVVM framework.

# What is a Component?

* A Component is basically a class that is defined for any visible element or controls on the page.
* TypeScript, a component is basically a TypeScript class decorated with an @Component() decorator.
* From an HTML point of view, a component is a user-defined custom HTML tag that can be rendered in the browser to display any type of UI element along with some business logic.
* A typical Angular application looks like a tree of components. The following diagram illustrates this concept.



# Component Structure

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

**export** **class** AppComponent {

  title = 'Welcome to Angular 8 Learning Series...';

}

# @Component Decorator

In Angular, to create any new component, we need to use the @Component decorator. @Component decorator basically classifies a TypeScript class as a component object. Actually, @Component decorator is a function that takes different types of parameters. In the @Component decorator, we can set the values of different properties to finalize or manipulate the behavior of the components. The most commonly used properties of the @Component decorator are:

1. **selector** – A component can be used by the selector expression. Many people treat components like a custom HTML tag because finally when we want to use the component in the HTML file, we need to provide the selector just like an HTML tag.
2. **template** – The template is the part of the component which is rendered in the browser. In this property, we can pass the HTML tags or code directly as inline code. Sometimes, we call this the inline template. To write multiple lines of HTML code, all code needs to be covered within the tilt (`) symbol.
3. **templateUrl** – This is another way of rendering HTML tags in the browser. This property always accepts the HTML file name with its related file path. Sometimes it is known as the external template. The use of this property is much better when we want to design any complex UI within the component.
4. **moduleId** – This is used to resolve the related path of template URL or style URL for the component objects. It contains the Id of the related modules in which the component is attached or tagged.
5. **styles / stylesUrls** – Components can be used in their own style by providing custom CSS, or they can refer to external style sheet files, which can be used by multiple components at a time. To provide an inline style, we need to use styles, and to provide an external file path or URL, we need to use styleUrls.
6. **providers** – In the real-life application, we need to use or inject different types of custom services within the component to implement the business logic for the component. To use any user-defined service within the component, we need to provide the service instance within the provider. Basically, the provider property is always allowed array-type value. So that we can define multiple service instance names that can be provided by comma separation within this property at a time.

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

@Component({

  selector: 'app-root',

  template: 'Welcome to Angular 8 Learning Series...'

})

**export** **class** AppComponent {

}

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

@Component({

  moduleId: module.id,

  selector: 'app-root',

  templateUrl: './app.component.html'

})

**export** **class** AppComponent {

  title = 'Welcome to Angular 8 Learning Series...';

}

### **Demo 1:**

1. **import** { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. template: '<h1>Component is the main Building Block in Angular</h1>'
6. })
7. **export** **class** AppComponent {
9. }

### **Demo 2: Apply Style into the Content**

1. **import** { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. template: '<h1>Component is the main Building Block in Angular</h1> <h2>Angular 8 Samples</h2>',
6. styles: ['h1{color:red;font-weight:bold}','h2{color:blue}']
7. })
8. **export** **class** AppComponent {
10. }

### **Demo 3: Use External Stylesheet File for Component**

1. h1{
2. color:red;
3. font-weight:bold;
4. font-size: 30px;
5. }
6. h2{
7. color:blue;
8. font-size: 20px;
9. }
11. p{
12. color:brown;
13. font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif;
14. }
15. **import** { Component } from '@angular/core';
17. @Component({
18. selector: 'app-root',
19. template: '<h1>Component is the main Building Block in Angular</h1> <h2>Angular 8 Samples</h2>',
20. styleUrls : ['./custom.css']
21. })
22. **export** **class** AppComponent {
24. }

### **Demo 4: Use External HTML File for Component Content**

1. **import** { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls : ['./custom.css']
7. })
8. **export** **class** AppComponent {
10. }

### **Demo 6: Nested or Parent-Child Component**

**child.component.ts**

1. **import** { Component } from '@angular/core';
3. @Component({
4. selector: 'child',
5. templateUrl: './child.component.html',
6. styleUrls : ['./custom.css']
7. })
8. **export** **class** ChildComponent {
10. }

**child.component.html**

1. <h2>It is a Child Component</h2>
2. <p>
3. A component is a Reusable part of the application.
4. </p>

**app.component.ts**

1. **import { Component } from '@angular/core';**
3. **@Component({**
4. **selector: 'app-root',**
5. **templateUrl: './app.component.html',**
6. **styleUrls : ['./custom.css']**
7. **})**
8. **export class AppComponent {**
10. **}**

**app.component.html**

1. <h1>Demostration of Nested Component **in** Angular</h1>
2. <h3>It is a Parent Component</h3>
3. <child></child>

Now include the child component into the app.module.ts file as below -

1. **import** { BrowserModule } from '@angular/platform-browser';
2. **import** { NgModule } from '@angular/core';
4. **import** { AppComponent } from './app.component';
5. **import** { ChildComponent } from './child.component';
7. @NgModule({
8. declarations: [
9. AppComponent,ChildComponent
10. ],
11. imports: [
12. BrowserModule
13. ],
14. providers: [],
15. bootstrap: [AppComponent]
16. })
17. **export** **class** AppModule { }