Angular framework architecture

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks are *NgModules.They* provide a compilation context for *components*.

An app always has at least a *root module* that enables bootstrapping, and typically has many more *feature modules*.

NgModules

Angular apps are modular and Angular has its own modularity system called *NgModules*. They can contain components, service providers, and other code files whose scope is defined by the containing NgModule. They can import functionality that is exported from other NgModules, and export selected functionality for use by other NgModules

Every Angular app has at least one NgModule class, the *root module*, which is conventionally named `AppModule` and resides in a file named `app.module.ts`. You launch your app by *bootstrapping* the root NgModule.

While a small application might have only one NgModule, most apps have many more *feature modules*. The *root* NgModule for an app is so named because it can include child NgModules in a hierarchy of any depth.

NgModule metadata

An NgModule is defined by a class decorated with `@NgModule()`. The `@NgModule()` decorator is a function that takes a single metadata object, whose properties describe the module. The most important properties are as follows.

- `declarations`: The components, *directives*, and *pipes* that belong to this NgModule.
- `exports`: The subset of declarations that should be visible and usable in the *component templates* of other NgModules.
- `imports`: Other modules whose exported classes are needed by component templates declared in *this* NgModule.
- `providers`: Creators of services that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level, which is often preferred.)
- `bootstrap`: The main application view, called the *root component*, which hosts all other app views. Only the *root NgModule* should set the `bootstrap` property.

EX:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [    ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Components

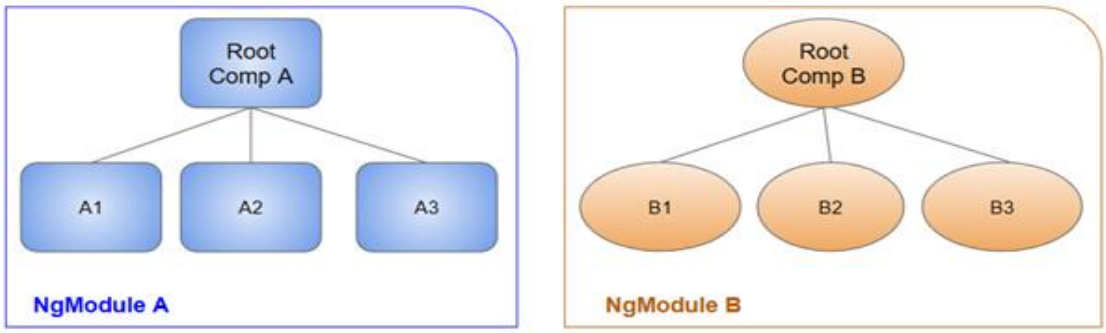
A *component* controls a patch of screen called a *view*. Once define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods.

Component metadata

The `@Component` decorator identifies the class immediately below it as a component class, and specifies its metadata. The metadata for a component tells Angular where to get the major building blocks that it needs to create and present the component and its view.

The metadata object of a component decorator has the following important properties.

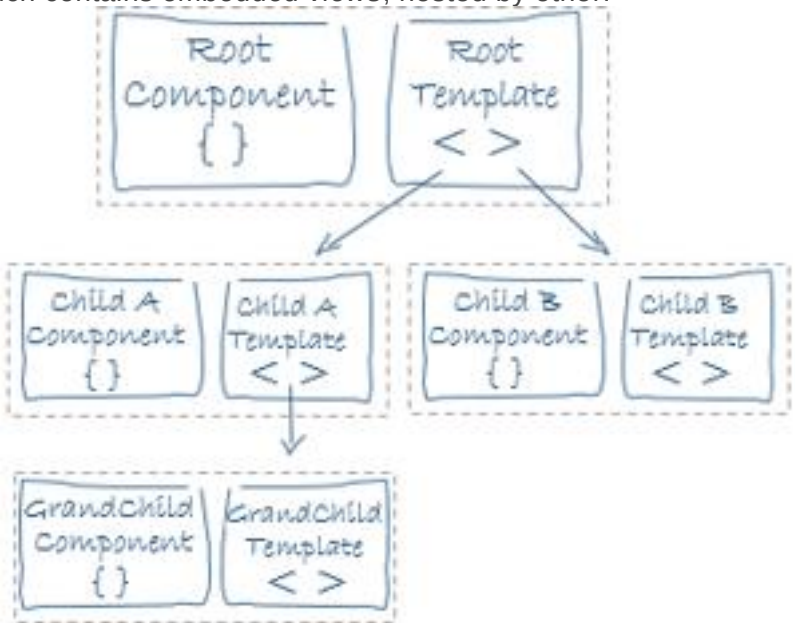
- **`selector`**: A CSS selector that tells Angular to create and insert an instance of this component wherever it finds the corresponding tag in template HTML.
- **`templateUrl`**: The module-relative address of this component's HTML template. This template defines the component's *host view*.
- **`styleUrls`**: An array of CSS files for that the component requires.



Templates and views

A template is a form of HTML that tells Angular how to render the component.

Views are typically arranged hierarchically, allowing you to modify or show and hide entire UI sections or pages as a unit. The template immediately associated with a component defines that component's *host view*. The component can also define a *view hierarchy*, which contains *embedded views*, hosted by other.



A template looks like regular HTML, except that it also contains Angular template syntax, which alters the HTML based on your app's logic and the state of app and DOM data.

Ex: Example of Angular component class

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})


export class AppComponent {
  title = 'AngularApp2020';
}
```