



Raising the bar

# Angular NgModule & Dependency Injection

# Mục tiêu

- Khai báo được các module tùy biến.
- Sử dụng được các module có sẵn.
- Phân biệt được các loại module.
- Trình bày được khái niệm Dependency Injection (DI) trong Angular.
- Triển khai được DI trong Angular.

# Angular NgModule

# Angular NgModule là gì?

- Angular app được chia thành các mô-đun, chúng được gọi là NgModule.
- NgModule có thể chứa các concept như: component, pipe, directive, service provider, các phần code khác mà định nghĩa trong nó.
- Chúng có thể import các chức năng khác từ các NgModule khác và có thể export một số chức năng mà nó quản lý để NgModule khác sử dụng.

# Angular NgModule là gì?

- Mỗi một Angular app đều có ít nhất một NgModule, nó được gọi là **root module** và thông thường được đặt tên là *AppModule*, trong file `app.module.ts`
- Bản thân Angular cũng chia các chức năng thành các module như: `ReactiveFormsModule`, `RouterModule`, `HttpClientModule`, etc

# Angular NgModule


- declarations: Dùng để khai báo các thành phần: components, directives and pipes mà nó thuộc về Module đó. Tất cả các loại class khác không thuộc ba nhóm trên đều không được cho vào array này.
- imports: các module khác mà module này cần sử dụng.
- exports: các chức năng mà module này có thể cho phép module khác sử dụng.
- providers: Khi làm việc với Dependency injection, chúng ta cần khai báo các Services cho Injector thực hiện việc nạp các dependencies.

# Angular NgModule

- bootstrap: Khi định nghĩa root NgModule, chúng ta cần nói cho Angular biết chương trình cần khởi tạo view nào (root component). Chúng ta chỉ nên set bootstrap ở root NgModule.

# Angular NgModule

```
@NgModule({  
  declarations: [AppComponent],  
  imports: [  
    BrowserModule,  
    ImgSliderModule,  
    ImageGalleryModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```



Import các NgModule khác



# Các NgModule hay sử dụng

NgModule	Import it from	Khi nào cần sử dụng
BrowserModule	@angular/platform-browser	Khi bạn muốn chạy app ở browser, chỉ import một lần duy nhất ở root NgModule
CommonModule	@angular/common	Khi bạn muốn dùng các directive, pipe như NgIf, NgFor, etc. Nếu bạn import BrowserModule thì không cần import module này
FormsModule	@angular/forms	Làm việc với Template-driven form
ReactiveFormsModule	@angular/forms	Làm việc với Reactive form
RouterModule	@angular/router	Làm việc với client routing
HttpClientModule	@angular/common/http	Làm việc để kết nối với server

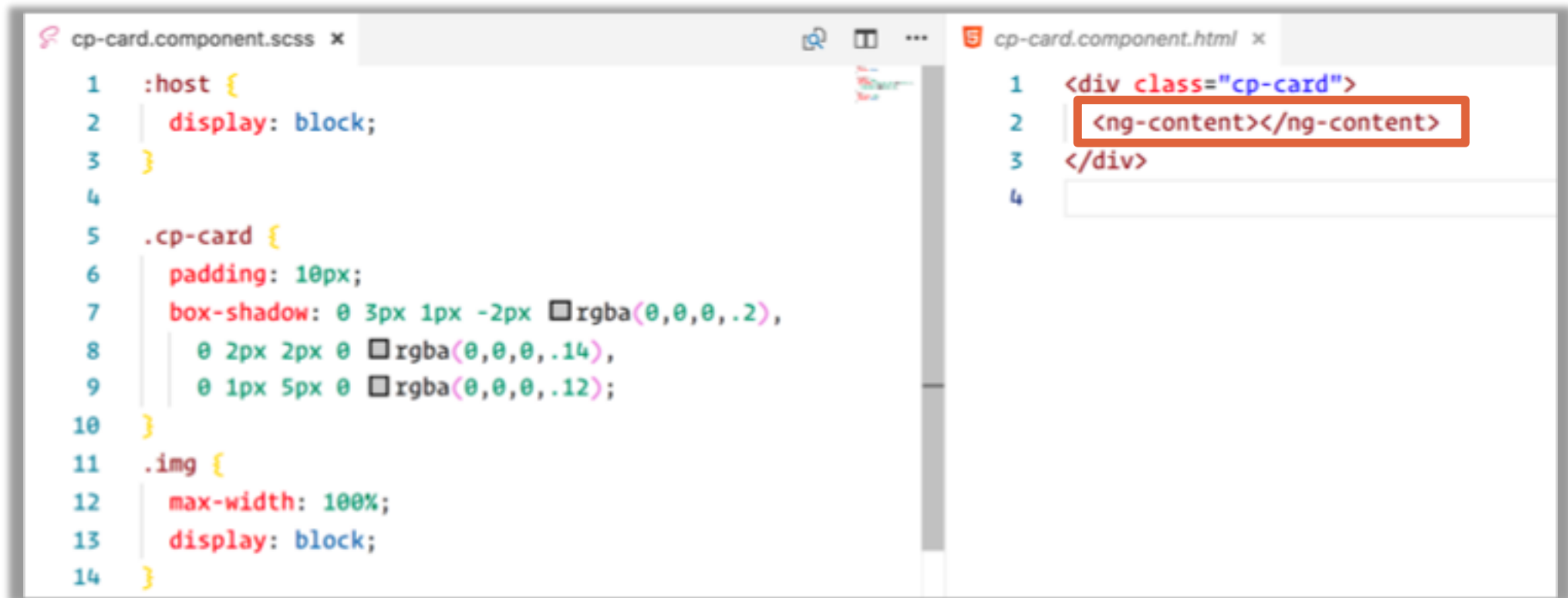
# Các loại NgModule

Loại NgModule	Mục đích
SharedModule	Đây là tên quy ước được đặt cho một NgModule mà module này quản lý các component, pipe, directive được sử dụng ở nhiều NgModule khác nhau trong Application.
CoreModule	Đây là tên quy ước được đặt cho một NgModule mà ở đó chúng ta khai báo các providers được sử dụng trong suốt app với một instance duy nhất của service đó. CoreModule được import duy nhất một lần ở root module, mà không import ở các module khác.
Feature Modules	Đây có thể là các UI module, module xử lý các nghiệp vụ, module cho việc route, etc. Nó có thể chứa các component, pipe, directive, service, etc mà module này cần dùng đến. Hầu hết các NgModule sẽ được liệt vào nhóm này.

# Content Projection

- Làm thế nào để tạo component mà nó sẽ là khung cơ bản cho một component khác có thể truyền template vào?
- Angular có một concept đó là content projection, bằng việc nhúng ng- content directive vào component.
- Content projection cho phép tái sử dụng component một cách hiệu quả mà không cần tạo nhiều component bị lặp lại content.

# Content Projection



The image shows a code editor with two files open: `cp-card.component.scss` and `cp-card.component.html`.

**cp-card.component.scss**

```
1 :host {  
2   display: block;  
3 }  
4  
5 .cp-card {  
6   padding: 10px;  
7   box-shadow: 0 3px 1px -2px rgba(0,0,0,.2),  
8     0 2px 2px 0 rgba(0,0,0,.14),  
9     0 1px 5px 0 rgba(0,0,0,.12);  
10 }  
11 .img {  
12   max-width: 100%;  
13   display: block;  
14 }
```

**cp-card.component.html**

```
1 <div class="cp-card">  
2   <ng-content></ng-content>  
3 </div>  
4
```

The `<ng-content></ng-content>` tag in the HTML file is highlighted with a red box, indicating the content projection mechanism.

# Content Projection

The image shows a code editor with two files open:

**content-projection.component.html**

```
1 <h2>Content Projection</h2>
2 <app-cp-card>
3   <h3>Card heading</h3>
4   <p>Card content</p>
5 </app-cp-card>
6
```

An orange box highlights the content inside the `<app-cp-card>` tag (lines 3-4), with an arrow pointing to a text box below it:

Content của app-cp-card

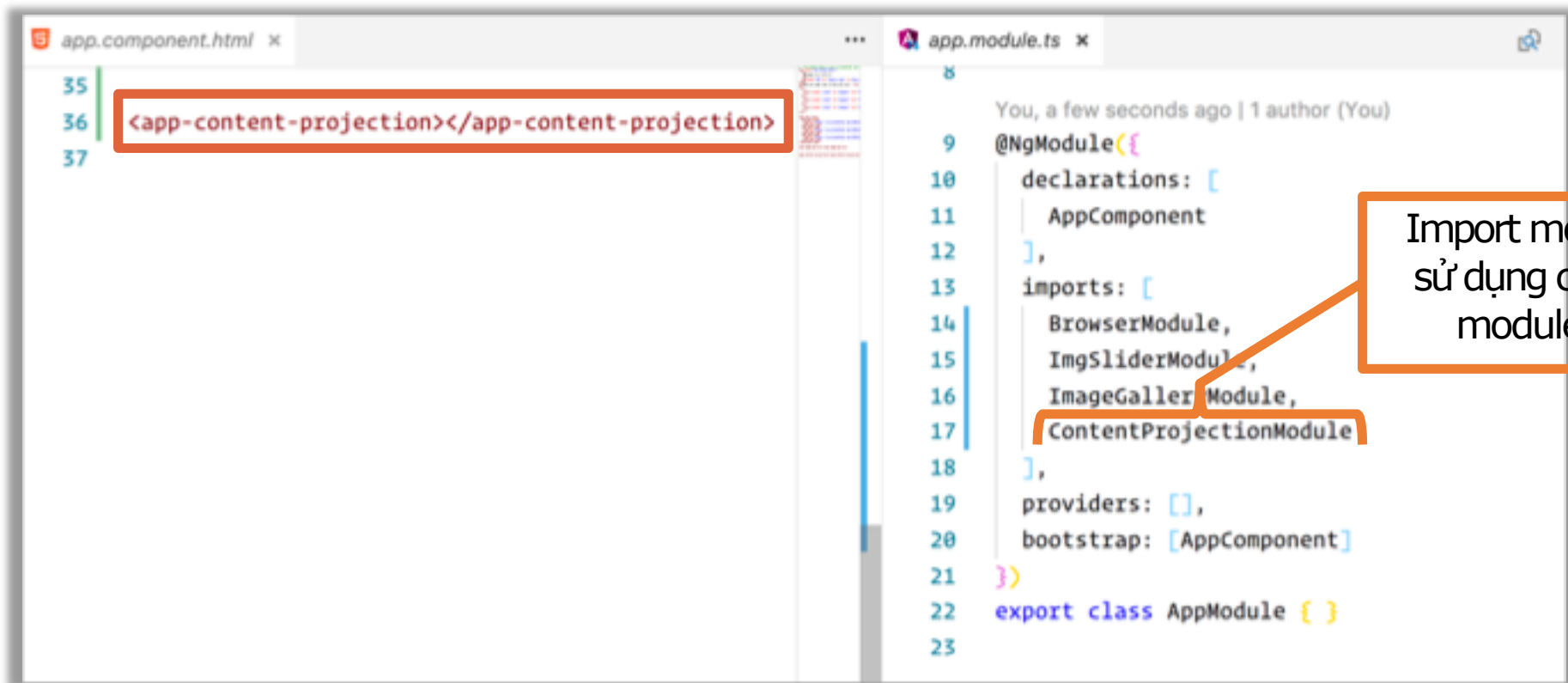
**content-projection.module.ts**

```
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { ContentProjectionComponent } from './content-projection';
4 import { CpCardComponent } from './cp-card/cp-card.component';
5
6 @NgModule({
7   imports: [
8     CommonModule
9   ],
10  declarations: [ContentProjectionComponent, CpCardComponent],
11  exports: [ContentProjectionComponent]
12 })
13 export class ContentProjectionModule { }
```

An orange box highlights the `exports` array in the `@NgModule` decorator (line 11), with a callout box next to it:

Export để module khác có thể sử dụng component này

# Content Projection



```
app.component.html x
35
36 <app-content-projection></app-content-projection>
37

...

app.module.ts x
8
You, a few seconds ago | 1 author (You)
9 @NgModule({
10   declarations: [
11     AppComponent
12   ],
13   imports: [
14     BrowserModule,
15     ImgSliderModule,
16     ImageGallerModule,
17     ContentProjectionModule
18   ],
19   providers: [],
20   bootstrap: [AppComponent]
21 })
22 export class AppModule { }
23
```

Import module để có thể sử dụng component mà module này export

# Content Projection

## Content Projection

**Card heading**

Card content

# Angular Dependency Injection

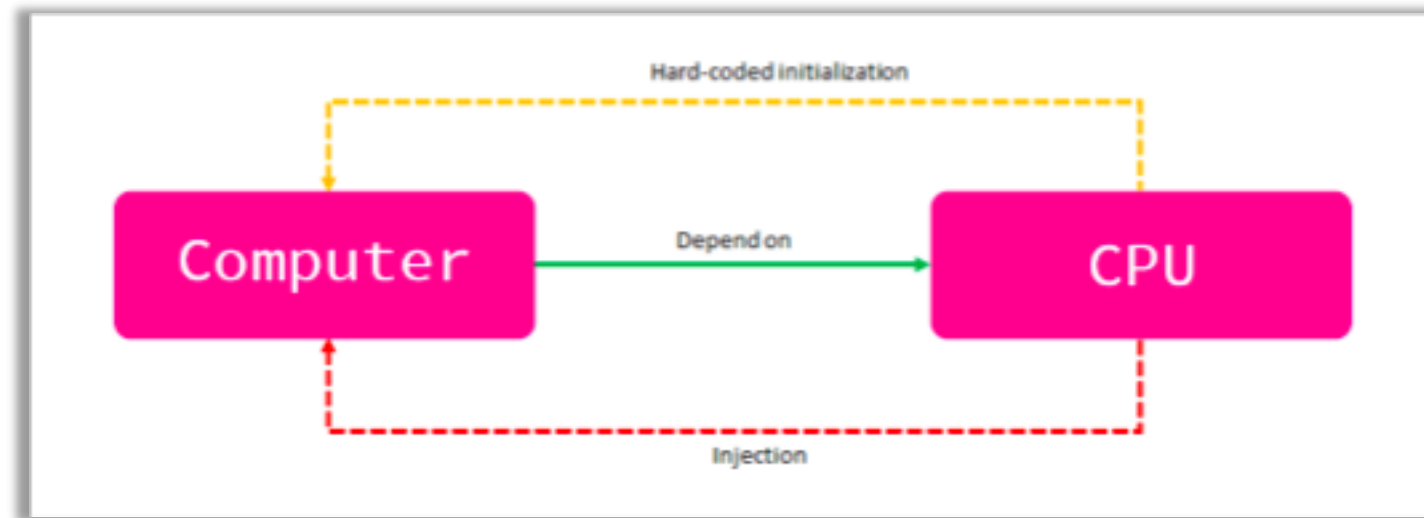


# Dependency Injection

- Dependency Injection (DI) được sử dụng rộng rãi trong Angular app.
- Dependencies là những services, objects mà class cần sử dụng đến. Class sẽ gửi yêu cầu đến nguồn bên ngoài để nạp cho nó, thay vì tự tạo. Giúp giảm tính kết dính giữa các class.
- Từ những service, router mà chúng ta đã học từ bài trước. Khi chúng ta đăng ký service, lúc này chúng ta đã đăng ký với DI framework của Angular. Angular DI sẽ chịu trách nhiệm khởi tạo và inject vào các class gửi yêu cầu đến nó.

# Dependency Injection

- Class Computer bị phụ thuộc vào class CPU.
- Class CPU là phụ thuộc của class Computer.



# Triển khai service

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

let defaultId = 1;

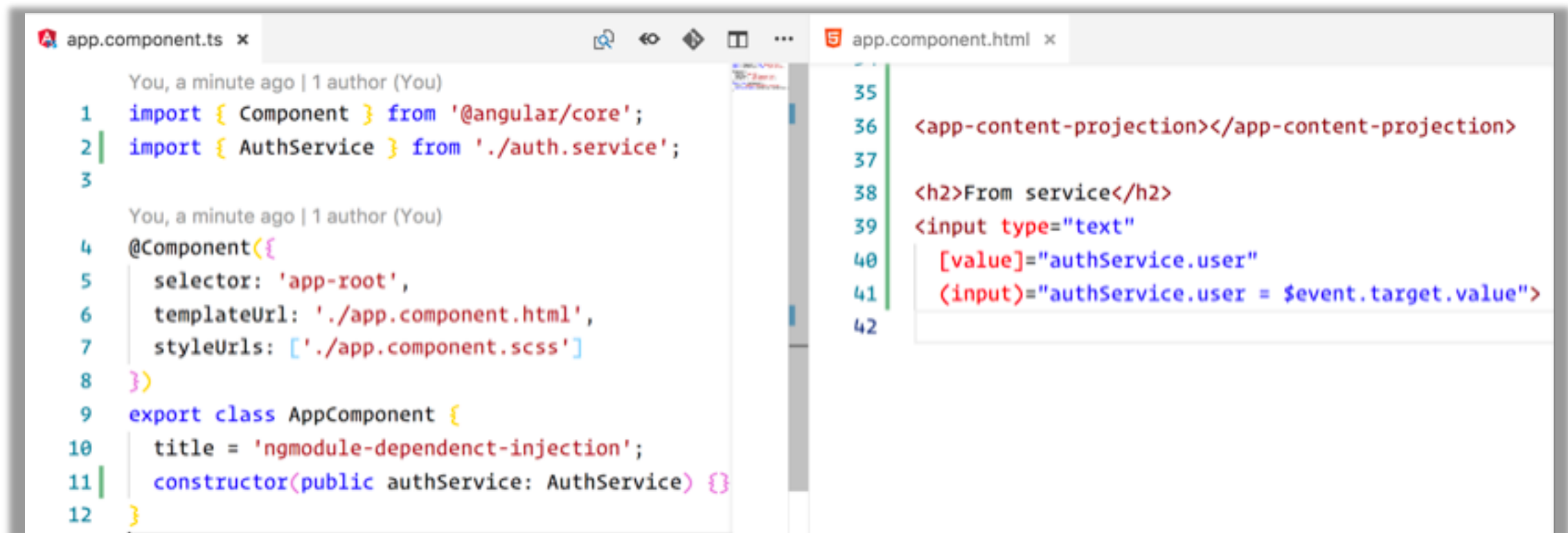
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  user: string = 'Default Id:' + defaultId++;
  constructor() { }
}
```

# Triển khai service

```
content-projection.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { AuthService } from '../auth.service';
3
4 @Component({
5   selector: 'app-content-projection',
6   templateUrl: './content-projection.component.html',
7   styleUrls: ['./content-projection.component.scss']
8 })
9 export class ContentProjectionComponent implements OnInit {
10
11   constructor(public authService: AuthService) { }
12
13   ngOnInit() {
14   }
15
16 }
```

```
content-projection.component.html x
1 <h2>Content Projection</h2>
2 <app-cp-card>
3   <h3>Card heading</h3>
4   <p>Card content</p>
5 </app-cp-card>
6
7 <p>
8   Auth Service: {{authService.user}}
9 </p>
10
```

# Triển khai service



The screenshot shows an IDE with two open files: `app.component.ts` and `app.component.html`.

**app.component.ts**

```
1 import { Component } from '@angular/core';
2 import { AuthService } from './auth.service';
3
4 You, a minute ago | 1 author (You)
5 @Component({
6   selector: 'app-root',
7   templateUrl: './app.component.html',
8   styleUrls: ['./app.component.scss']
9 })
10 export class AppComponent {
11   title = 'ngmodule-dependenct-injection';
12   constructor(public authService: AuthService) {}
13 }
```

**app.component.html**

```
35
36 <app-content-projection></app-content-projection>
37
38 <h2>From service</h2>
39 <input type="text"
40   [value]="authService.user"
41   (input)="authService.user = $event.target.value">
42
```

# Triển khai service

- Lúc này mặc dù ở 2 component khác nhau, nhưng dữ liệu được đồng nhất với nhau, vì chúng cùng sử dụng một instance của auth service

## Content Projection

**Card heading**

Card content

Auth Service: Default Id:1

**From service**

Default Id:1

## Content Projection

**Card heading**

Card content

Auth Service: Bob

**From service**

Bob

# Provider in-depth

- Ngoài cách khai báo service với `providedIn` trong metadata của `Injectable` decorator, chúng ta còn có thể khai báo ở `level module` hoặc `component`.

# Provider in-depth

@Injectable()

```
export class AuthService {}
```



# Provider in-depth

- useClass: cách này thường có dạng viết tắt

```
@NgModule({  
  providers: [AuthService],  
})
```

```
export class AppModule { }
```

# Provider in-depth

- useClass: cách này thường có dạng viết tắt

```
@NgModule({  
  providers: [  
    {provide: AuthService, useClass: AuthService}  
  ],  
})  
export class AppModule { }
```

# Provider in-depth

- `useValue`: cách này nếu chúng ta có sẵn một value, hoặc một cách nào đó module đó được cung cấp 1 value, có thể bạn sẽ thấy quen thuộc với config của Router

```
providers: [  
  {  
    provide: 'API_ENDPOINT',  
    useValue: 'http://api.example.com'  
  }  
]
```

# Provider in-depth

- useValue:

```
/**
 * @description
 *
 * Registers routes.
 *
 * ### Example
 *
 * ```
 * @NgModule({
 *   imports: [RouterModule.forChild(ROUTES)],
 *   providers: [provideRoutes(EXTRA_ROUTES)]
 * })
 * class MyNgModule {}
 * ```
 */
export function provideRoutes(routes: Routes): any {
  return [
    {provide: ANALYZE_FOR_ENTRY_COMPONENTS, multi: true, useValue: routes},
    {provide: ROUTES, multi: true, useValue: routes},
  ];
}
```

# Provider in-depth

- useExisting: sử dụng một tên khác cho một provider đã có

```
providers: [  
    {provide: 'API_URL', useExisting: 'API_ENDPOINT'}  
]
```

# Provider in-depth

- useExisting: sử dụng để tạo validator directive trong Angular form chẳng hạn

```
export const REQUIRED_VALIDATOR: StaticProvider = {
  provide: NG_VALIDATORS,
  useExisting: forwardRef(() => RequiredValidator),
  multi: true
};

export const CHECKBOX_REQUIRED_VALIDATOR: StaticProvider = {
  provide: NG_VALIDATORS,
  useExisting: forwardRef(() => CheckboxRequiredValidator),
  multi: true
};
```

# Provider in-depth

- useFactory: sử dụng để khởi tạo các value một cách tường minh. Hoặc bạn cần tạo một instance của một service có tham số của hàm tạo là kiểu primitive chẳng hạn.

```
{  
  provide: 'some-token',  
  useFactory: function() {  
    return Math.random();  
  }  
}
```

# Overrides Provider

- Khi có nhiều providers có cùng giá trị của key **provide** và không sử dụng config multi: true thì provider nào thêm vào sau cùng sẽ win.

```
provides:  [  
    {provide: 'API_URL', useValue: 'abc.com'},  
    {provide: 'API_URL', useExisting: 'api.com'}  
]
```

- Giá trị của token API\_URL lúc này sẽ là api.com



# NgModule Providers

- Khi bạn đăng ký service ở providers của NgModule decorator, tất cả component không đăng ký lại service sẽ nhận được cùng một instance như ví dụ trước.

## Content Projection

**Card heading**

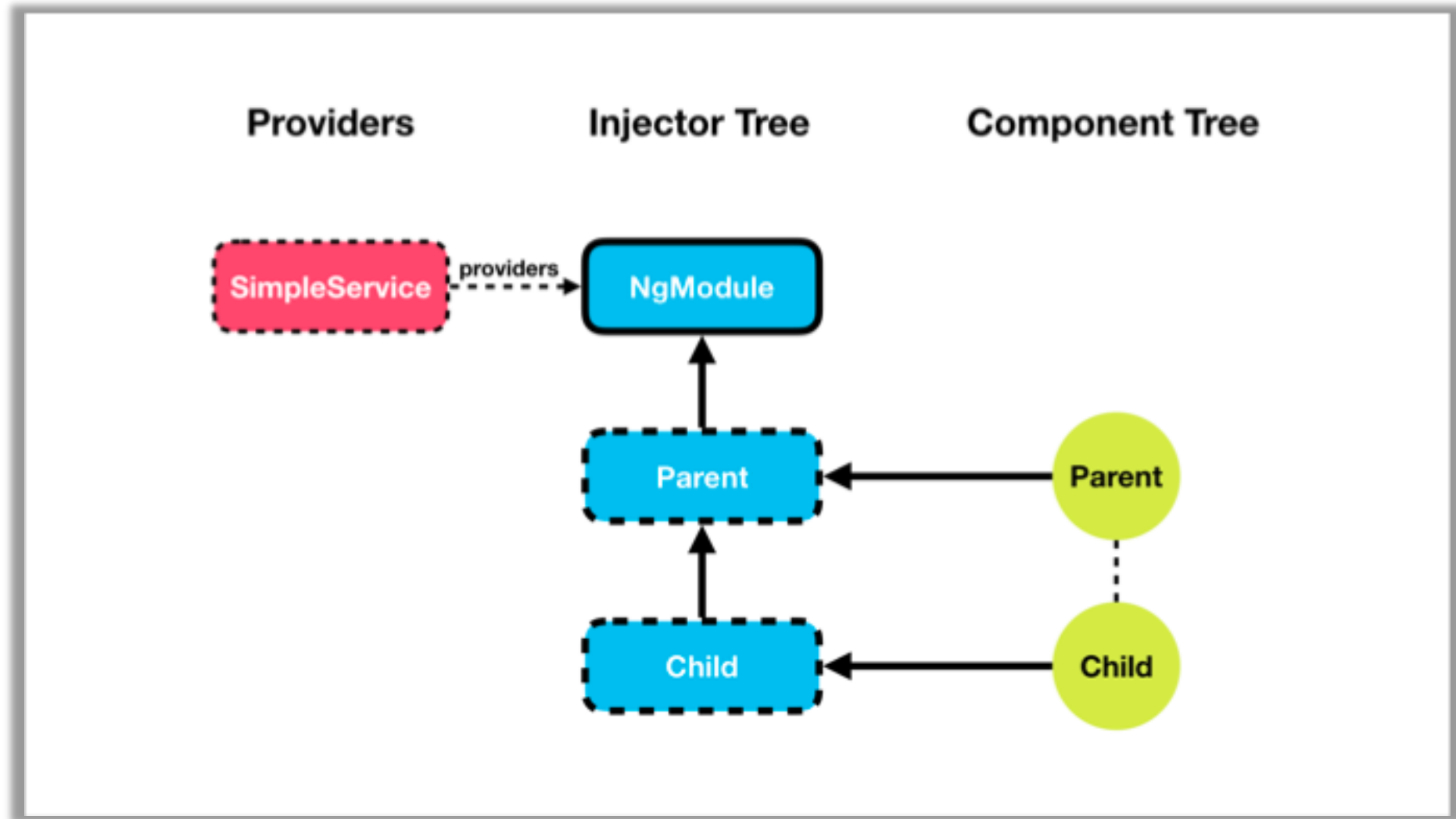
Card content

Auth Service: Default Id:1

## From service

Default Id:1

# NgModule Providers



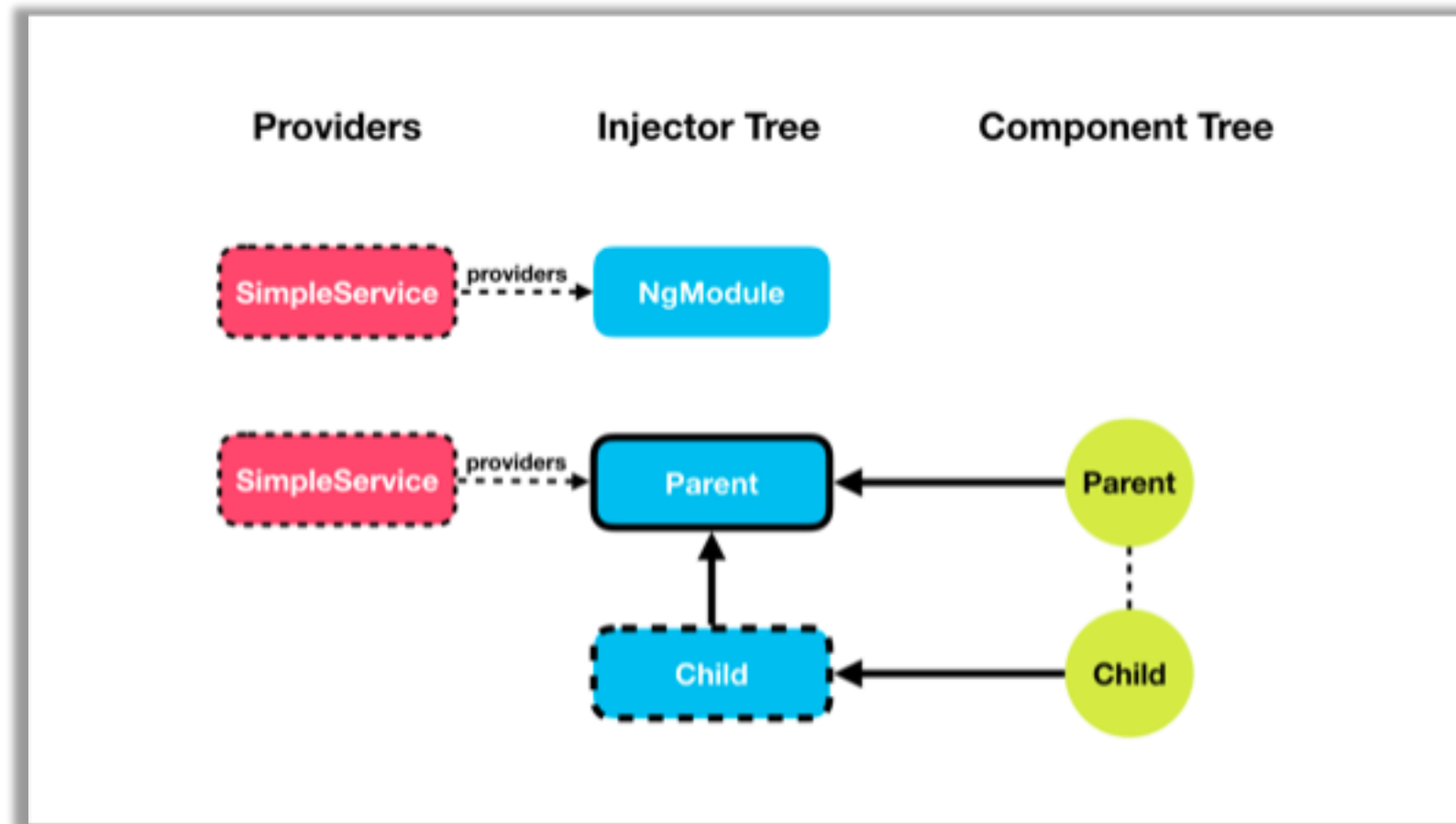
# NgModule Providers

- Khi một class request một dependency, DI system sẽ tìm ở cây injector, nếu không tồn tại sẽ tìm ngược lên parent, để tìm ra phần tử thỏa mãn, sau đó nạp vào cho class.
- Trong trường hợp với app của chúng ta, chúng ta đã khai báo service ở root NgModule, nên mặc dù ContentProjectionModule chúng ta không đăng ký cho service, nhưng khi class ContentProjectionComponent request auth service, thì nó và AppComponent sẽ cùng sử dụng một instance của auth service.

# Component Providers

- Khi bạn khai báo một service ở level component A, lúc này tất cả child của nó (nếu không khai báo lại service) sẽ sử dụng cùng một instance.
- Khi có 2 instances của component A sẽ có 2 instances của service tương ứng.

# Component Providers



# Component Providers

```
@Component({  
  selector: 'app-content-projection',  
  providers: [AuthService]  
})  
export class ContentProjectionComponent {  
  constructor(public authService: AuthService) { }  
}
```

# Component Providers

**Content Projection**

**Card heading**

Card content

Auth Service: Default Id:2

**From service**

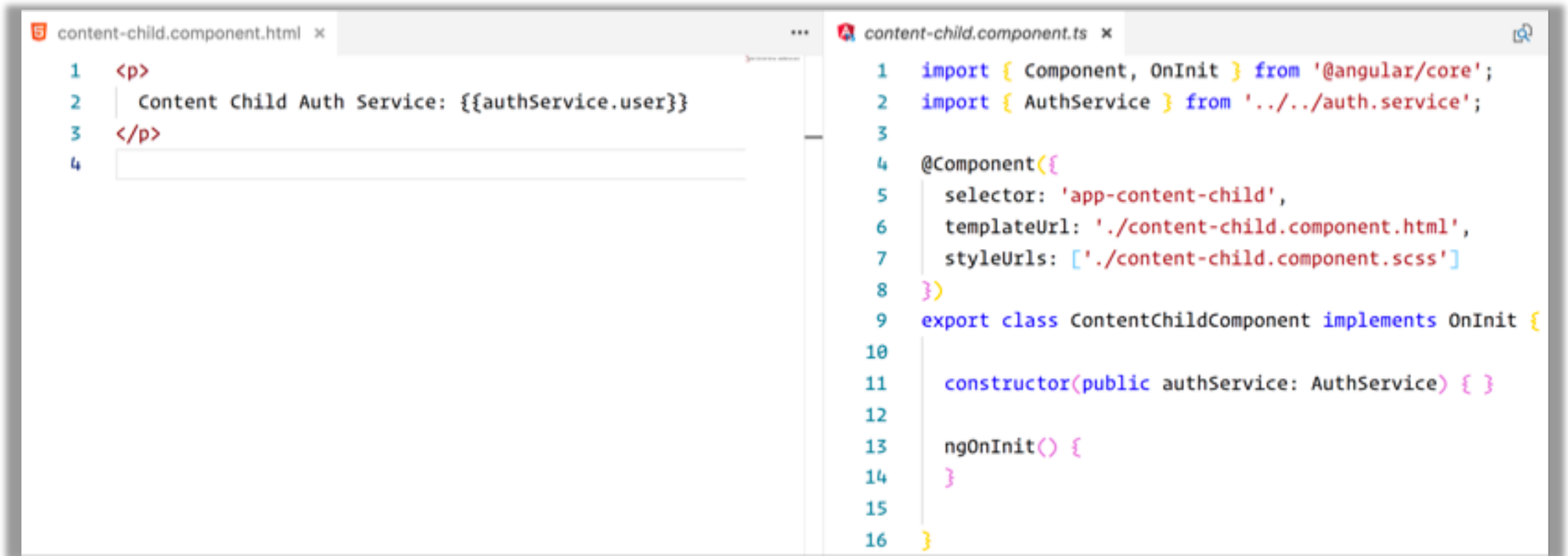
Bob

# Component ViewProviders

- Tạo mới component theo lệnh dưới đây
- `ng g c content-projection/content-child`
- Update content-projection component để sử dụng viewProviders.
- Khi bạn sử dụng viewProviders cho component A, thì những gì là child view của A sẽ sử dụng cùng một instance của một service nếu nó không khai báo lại service đó.
- Nhưng các phần tử là content được truyền vào, sẽ không sử dụng cùng một instance service đó.



# Component ViewProviders



```
content-child.component.html x
1 <p>
2   Content Child Auth Service: {{authService.user}}
3 </p>
4

content-child.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { AuthService } from '../auth.service';
3
4 @Component({
5   selector: 'app-content-child',
6   templateUrl: './content-child.component.html',
7   styleUrls: ['./content-child.component.scss']
8 })
9 export class ContentChildComponent implements OnInit {
10
11   constructor(public authService: AuthService) { }
12
13   ngOnInit() {
14   }
15
16 }
```

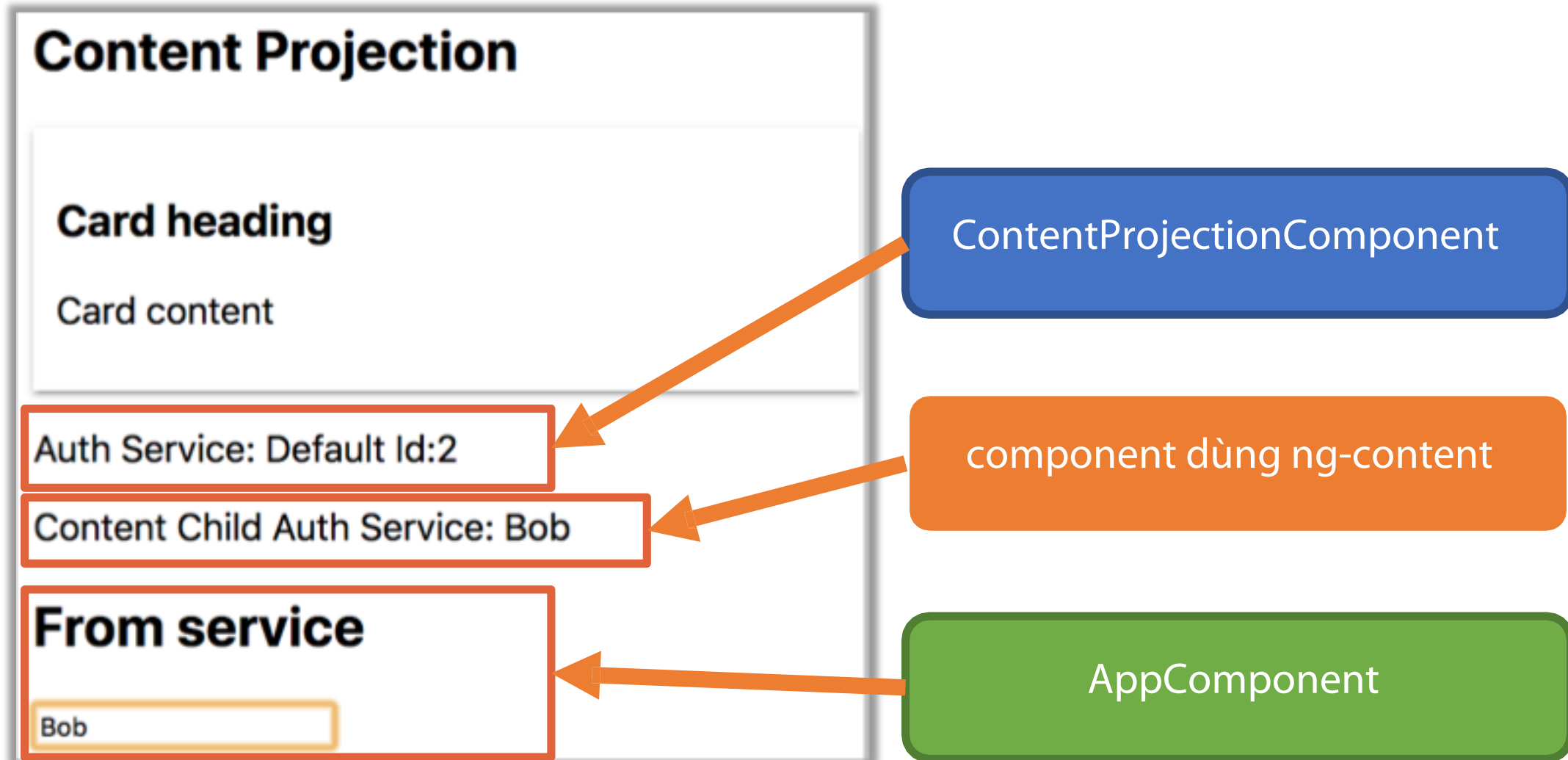
# Component ViewProviders

```
@Component({  
  selector: 'app-content-projection',  
  viewProviders: [AuthService]  
})  
export class ContentProjectionComponent {  
  constructor(public authService: AuthService) { }  
}
```

# Component ViewProviders

```
<app-content-projection>  
  <app-content-child></app-content-child>  
</app-content-projection>
```

# Component ViewProviders





Raising the bar