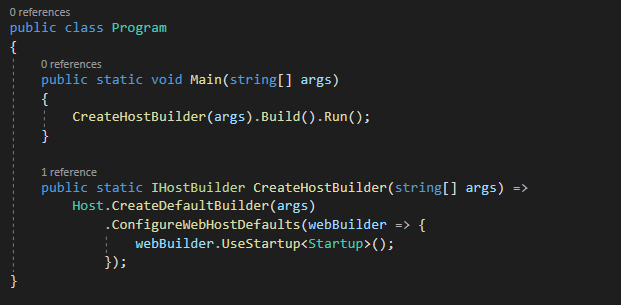
# Backend

## Tạo project và config

3 file để config: Program.cs, Startup.cs, and appsetting.json. Chúng config cho web application bao gồm các modules và middlewares, compilation settings, and publishing rules

Program.cs



Tạo WebHostBuilder, cái mà sẽ host web application của ta.

web host chịu trách nhiệm cho application startup và quản lý lifetime, web server chịu trách nhiệm việc chấp nhận HTTP requests. nhiệm vụ của Host bao gồm việc chắn chắn rằng services của application và server thì luôn sẵn sàng và được config một cách thích hợp

### Host

ASP.NET Core app builds một host, host này đóng gói tất cả app’resources

CreateDefaultBuilder và ConfigureWebHostDefaults methods config một host với các options mặc định (chỗ này chính là chỗ tạo host)

* Sử dụng Kestrel làm web server (cái mà nhận request) và enable IIS integration
* Load config từ appsetting.json, appsettings.{Environment Name}.json, environment variables, command line arguments, và các config khác
* gửi logging tới console

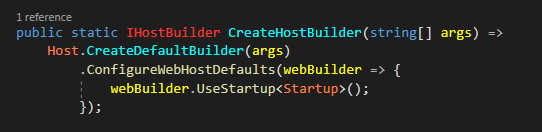
## Detail về host, web server

### The Startup class

Đây là nơi

* config các services được yêu cầu
* miêu tả xử lý pipeline request, một tập hợp các component xử lý middleware

#### Host



host đóng gói tất cả app’s resources như:

* HTTP server implementation (Kestrel, IIS,..)
* Middleware
* Logging
* DI
* Config

Có 2 host khác nhau:

* .NET Generic Host
* ASP.NET Core Web Host

.NET Generic Host thì được đề xuất. ASP.NET Core Web Host thì chỉ cho việc tương thích ngược.

CreateDefaultBuilder và ConfigureWebHostDefaults là 2 phương thức gọi để config một host với một tập các options default, và ở đây chúng là

* Sử dụng Kestrel là web server và tích hợp IIS
* Load config từ appsetting.json, appsetting.{Environment Name}.json, environment variables, command line arguments, và các config khác
* Send logging output to the console and debug provides

Một Generic Host cho phép các kiểu khác của các apps sử dụng trên framework extensions, như logging, DI, config, app lifetime management

#### Server

ASP.NET Core app sử dụng một HTTP server implementation để lắng nghe HTTP request.

Với Linux ASP.NET Core cung cấp Kestrel – một HTTP server đa nền tảng, ta thường chạy một reverse proxy server đứng trước kestrel là Nginx hoặc Apache

##### Configuration

ASP.NET Core cung cấp một configuration framework, nhận vào một cặp name-value. Built-in configuration cung cấp một loạt các config sẵn như .json files, .xml files, environment variables.

Bởi mặc định, ASP.NET Core apps được config để đọc từ appsetting.json, environment variables, the command line. Khi mà app’s config được load thì các giá trị từ environment variables sẽ ghi đề các giá trị từ appsetting.json

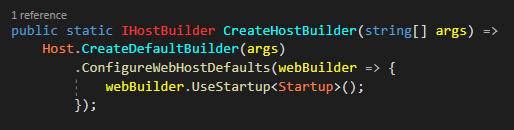
##### Environment

Development, Staging, and Production. Chỉ định dùng là ASPNETCORE\_ENVIRONMENT environment variable. ASP.NET Core đọc giá trị môi trường này từ app startup và lưu trữ giá trị này trong IWebHostEnvironment implementation

### Startup class

* Bao gồm ConfigureServices method để config app’s services. Một service là một component có thể tái sử dụng, nó cung cấp tính năng cho app. Services được đăng ký trong ConfigureServices và được sử dụng thông qua DI hoặc ApplicationServices (cung cấp khả năng truy cập tới app’s service container)
* Bao gồm một Configure method để tạo xử lý pipeline request của app.

Startup class được xác định khi host của ứng dụng được xây dựng



host cung cấp các services cái mà có sẵn để dụng cho Startup class constructor. Các services khác được thêm thông qua ConfigureServices. Cả host và app services thì đều có sẵn để dùng trên Configure hoặc toàn bộ app

Phần lớn các services không có sẵn cho đến khi được gọi bởi Configure

#### ConfigureServices method

* Optional
* Được gọi bởi host trước Configure method để configure các services của app
* Là nơi mà configuration options là set by convention

host có lẽ config nhiều services trước khi Startup methods được gọi.

Đối với các tính năng có yêu cầu substantial setup, chúng ta có Add{Service} extension method trên IServiceCollection, như AddDbContext, AddDefaultIdentity...

#### Configure method

Được sử dụng để xác định cách mà app phản hồi HTTP requests. The request pipeline được config bởi việc thêm middleware components tới IApplicationBuilder instance. IApplicationBuilder có sẵn cho Configure nhưng nó không được đăng ký trong service container. Hosting tạo một IApplicationBuilder và truyền trực tiếp nó tới Configure

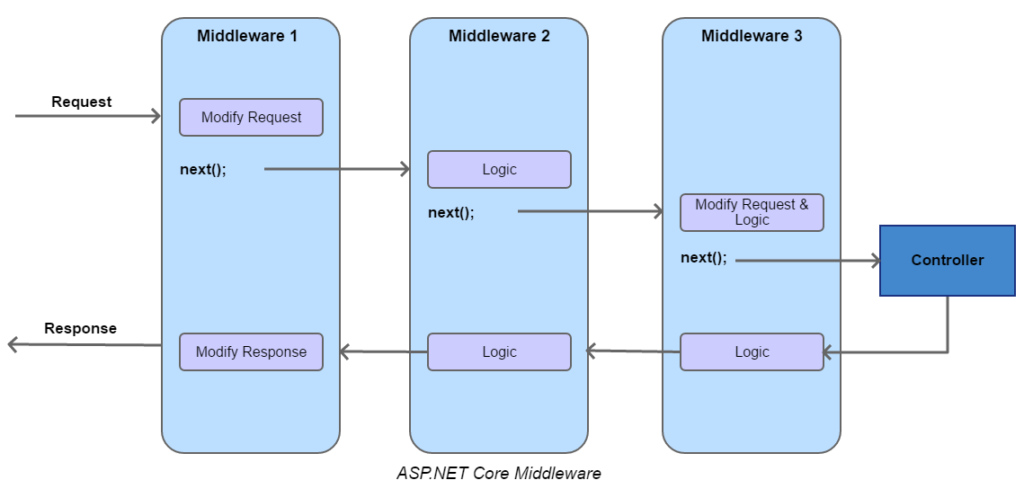
Mỗi “Use” extension method thêm một hoặc nhiều hơn middleware components tới request pipeline. Ví dụ, UseStaticFiles configure middleware để phục vụ static files

Mỗi middleware component trong request pipeline thì chịu trách nhiệm cho việc invoke component tiếp theo trong pipeline, các services đã được add trong ConfigureServices đều có thể được inject vào trong Configure

### ASP.NET Core Middleware

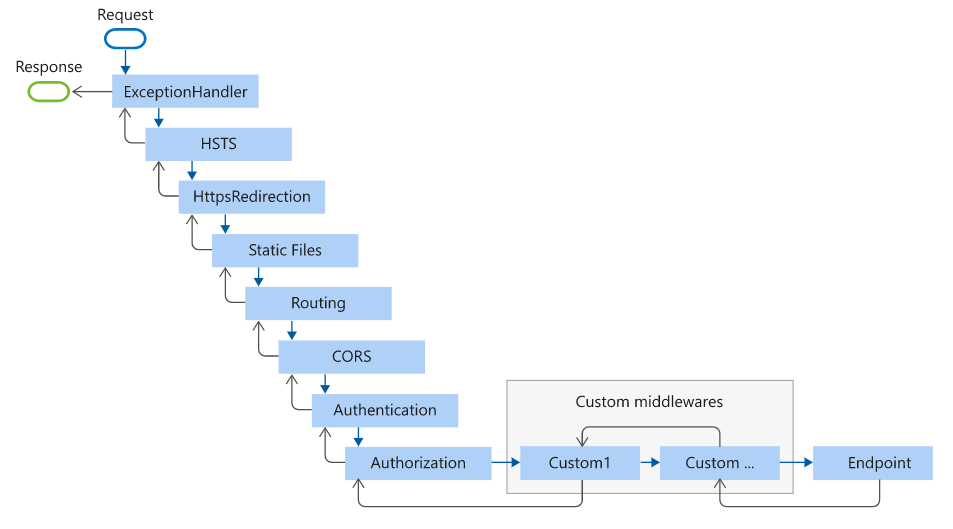
* Chọn liệu có pass request tới next component trong pipeline
* Có thể thực hiện công việc trước và sau next component trong pipeline

Request delegates: là phần xử lý được sử dụng trong Run, Map hoặc Use, nó có thể là anonymous method hoặc method được định nghĩa trong một class nào đó. Và request delegate cũng gọi là middleware hay middleware component. Mỗi component bên trong request pipeline chịu trách nhiệm cho việc gọi component tiếp theo hoặc short-circuiting the pipeline.



Mỗi delegate có thể thực thi hành động trước và sau next delegate. Exeption-handling delegates có thể được gọi sớm trong pipeline, vì vậy bạn có thể bắt exeptions cái mà gặp ở giai đoạn sau của pipeline

#### Middleware order





Thứ tự tiêu chuẩn theo code trên

* Middleware không được thêm khi mà việc tạo một new web app với users accounts is commented
* Không phải nhất thiết mọi middlware cần theo thứ tự kia, nhưng với UseCors, UseAuthentication và UseAuthorization phải theo order

middleware common cho các kịch bản apps nói chung

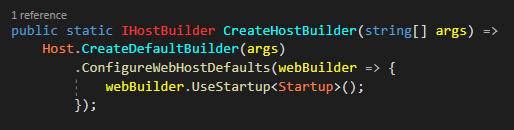
1. Exeption/error handling

* IsDevelopment – UseDeveloperExeptionPage
* Production – UseExeptionHandler

1. HTTPS Redirection chuyển hướng HTTP request tới HTTPS
2. Cookie Policy Middleware (UseCookiePolicy) tuân theo app tới EU General Data Protection Regulation
3. Routing Middleware (UseRouting) để route (định tuyến) requests
4. Authentication Middleware (UseAuthentication) cố gắng để xác thực user trước khi cho phép chúng truy cập tới tài nguyên được bảo vệ
5. Authorization Middleware cấp quyền cho user để truy cập tới tài nguyên được bảo vệ
6. Session Middleware thiết lập và duy trì session trạng thái. Mếu app sử dụng session state, gọi nó trước Cookie và sau MVC Middleware
7. Endpoint Routing Middleware

### Host

Khi một host start, nó gọi IHostedService.StartAsync trên mỗi thực thi của IHostedService cái mà tìm thấy trong DI container. Trong một web app, mỗi một IHosterService là một web service cái mà bắt đầu một HTTP server implementation

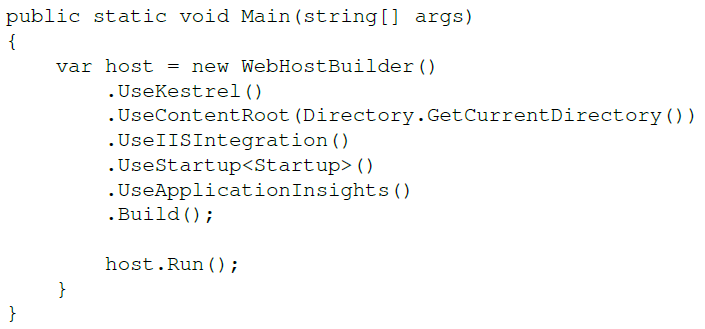


Default builder settings

The CreateDefaultBuilder method:

* Sét the content root to the path retured bt GetCurrentDirectory
* Loads host config from:
* Environment variables prefix with DOTNET\_
* Command-line argument
* Loads app config from:
* appsetting.json
* appseting.{Environment}.json
* Secret Manager when app runs in the Development environment
* Environment variables
* Command-line arguments
* Add the following logging provides
* Console
* Debug
* EventSource
* EventLog
* Enables scope validation and dependency validation when the environment is Development

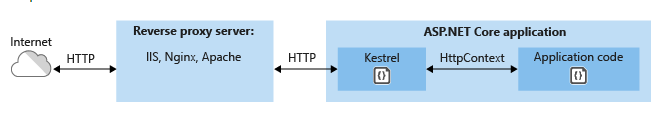
The ConfigureWebHostDefaults

* Load host config from environment variables prefix ASPNETCORE\_
* Thiết lập Kestrel là web server và config nó sử dụng các cofig hosting của app.
* Thêm Host Filtering middleware
* Thêm Forwarded Headers middleware nếu ASPNETCORE\_FORWARDEDHEADERS\_ENABLED bằng true
* cho phép IIS tích hợp
* Trong .NET Core 1.x ta có Program.cs như sau:
* 
* Với .NET Core 2.x và 3.x ta vẫn có như thê nhưng mọi thứ được gói gọn lại trong một method là **CreateHostBuilder()**

### Server

#### Kestrel

Kestrel là default web server được chỉ định bởi ASP.NET Core project template



#### ASP.NET Core server infrastructure

##### Configuration

Config trong ASP.NET Core được thực hiện bằng việc sử dụng một hoặc nhiều hơn config providers. Config providers đọc config data từ key-value sử dụng một vài config sources:

* setting files, giống như appsettings.json
* Environment variables
* Azure Key Vault
* Azure App Config
* Command-line arguments
* Custom providers, installerd or created
* Directory files
* In-memory .NET objecsts

.... Tạm thế đã về phần này

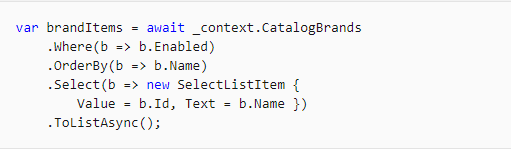
## Lý thuyết về query data với ASP.NET Core Apps

Các giai đoạn query.

Trước query – Tạo Query với Linq – Execute IQuerybable (đã tạo với linq) – Tracking của Context với entities đã đạt được – SaveChange để lưu lại các thay đổi đối với các entities nó đang tracking.

Client vs Server Evaluation – EF Core cố gắng để query trên server nhiều nhất có thể (tức là trên database). EF Core convert các phần của query vào bên trong các parameters, cái có thể làm trên client. Toàn bộ query (cùng với các parameters đã generated) thì được trao cho database provider để truy vấn trên server database. EF Core hỗ trợ tầng client phía top level (có nghĩa ở level người dùng sau query trên database khi mà đã có dữ liệu trả về nơi gửi lệnh query) (bản chất ở đây là method Select()). Nếu mà client (nơi ta code) không thể biên dịch được sang cho server để nó query, EF Core sẽ lấy toàn bộ dự liệu từ server với các query còn lại. Nếu EF Core phát hiện bất cứ một expression nào trong top-level, cái đó sẽ không được dịch sang server

Giả dụ ta có một đoạn query với Linq như sau



Với đoạn trên, từ Where, OrderBy, Select đều là các option của linq mà ta áp vào để query data. Quan trọng nhất trong câu lệnh trên là ToListAsync, nó làm nhiệm vụ là execute query ngay lập tức. Nếu không có thằng ToListAsync, câu lệnh sẽ được gán vào một IQuerynable<SelectedItem> , và thằng này sẽ không được executed cho đến khi nó được làm rõ. Với thằng này, nó cho phép query EF Core sẽ xây dựng và xa hơn là chỉnh sửa, nhưng cũng có thể bị lỗi trong runtime.

### Tracking

Đây là phần khá quan trọng ta cần nói chi tiế, bên trên là nói về thời điểm mà provider sẽ execute query còn ở đây sẽ là sau khi query thì Context sẽ tương tác với entities đã đạt được như thế nào

EF Core theo dõi thay đổi trên các entities mà nó fetch từ provider. Và chúng ta chỉ cần SaveChanges là nó sẽ lưu lại thay đổi. Để loại bỏ việc tracking này đơn giản là ta thêm method AsNoTracking.

Để chi tiết nhất, đọc tài liệu EF Core phần Database Queries và Modifing Data with EF Core

### Query Type

Có vài loại query là

* Eager loading (sử dụng Include) – Nó sẽ query luôn cả navigation property ngay cùng câu lệnh. Ưu điểm là chỉ request đến database một lần, nhược điểm là ta phải load toàn bộ cả cái mà ta không cần
* Explicit loading (sử dụng Entry) – thằng này sử dụng để query navigation property sau lần đầu tiên ta query mà không bao gồm navigation property. Ưu điểm là ta chỉ cần query cái ta muốn, nhược điểm là sẽ request đến database nhiều lần
* Raw sql query

Chốt lại

1. Hiểu quá trình, điểm thời gian của query
2. nắm tốt 2 kiểu query là eager load và explicit
3. Nắm rõ tracking

## Authentication

Authentication là quá trình xác định danh tính user. Authorization là quá trình xác định liệu một user có khả năng truy cập đến tài nguyên. Trong ASP.NET Core, authentication được xử lý bởi IAuthenticationSerive, service này thì được sử dụng bởi authentication middleware. authentication service đã đăng kí để hoàn thành các hành động liên quan đến xác thự như

Xác thực một user

Phản hồi khi một user chưa được xác thực đang cố để truy cập đến tài nguyên hạn chế

Các xử lý xác thực đã đăng ký và config chúng được gọi là “schemes” (tức là xử lý và config với các user đã xác thực)

schemes được xác định bởi việc đăng ký authentication-service trong Startup.ConfigureServices

* gọi extension method scheme-specific sau khi mà ta gọi tới services.AddAuthentication(như là AddJwtBearer hoặc AddCookie). Các extension method này sử dụng AuthenticationBuilder.AddScheme để đăng ký schemes với các setting thích hợp
* ít khi gặp trường hợp gọi AuthenticationBuilder.AddScheme một cách trực tiếp

Ví dụ, đoạn code sau đăng ký authentication-service và handler cho cookie và JWT

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options => Configuration.Bind("JwtSettings", options))

.AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, options => Configuration.Bind("CookieSettings", options));

method AddAuthentication có parameter là JwtBearerDefault.AuthenticationScheme là tên scheme mặc định được sử dụng khi mà ta không có một scheme yêu cầu.

Nếu có nhiều schemes được sử dụng, chính sách authorization có thể xác định sử dụng authentication-scheme cái mà chúng phụ thuộc để xác thực user. Trong ví dụ trên, cookie authen-scheme có thể được sử dụng bởi việc chỉ định tên của nó (mặc định là CookieAuthenticationDefaults.AuthenticationScheme, và thông quan một tên khác có thể được cung cấp khi gọi AddCookie).

### Authentication Concepts

#### Authentication scheme

Scheme là một cơ chế hiệu quả cho việc nhắm tới authentication, challenge and forbid behaviors của handler có liên quan. Ví dụ, một authorization policy có thể sử dụng scheme names để xác định scheme sẽ được sử dụng xác nhận user. Khi config authentication, nó chỉ định chung tới default scheme. default scheme được sử dụng trừ phi bị yêu cầu một scheme xác định.

#### Authentication handler

Một authentication handler:

* Is a type that implements the behavior of a scheme
* Được dẫn xuất từ IAuthenticationHandler hoặc AuthenticationHandler<TOptions>
* Has the primary responsibility to authenticate users

Trên cơ sở scheme’s config và incoming request context, handlers:

* Xây dựng các objects Authentication Ticket đại diện user’s identity nếu việc xác thực thành công
* Trả về ‘no result’ hoặc ‘failure’ nếu việc xác thực thất bại
* Có các method cho challenge và forbid actions khi các users cố tình truy cập tài nguyên
* Họ không được cấp quyền để truy cập (forbid)
* Khi họ chưa xác thực (challenge)

#### Authenticate

Chịu trách nhiệm cho việc xây dựng user’s identity dựa trên context được yêu cầu. Nó là một action của scheme. Trả về AuthenticateResult chỉ ra là có xác thực thành công hay k.

#### Challenge

Một authentication challenge được triệu hồi bởi Authentication khi mà user chưa được xác thực mà request tới endpoint cần việc xác thực. Khi một user nặc danh request một tài nguyên bị hạn chế. Authentication invoke challenge bằng việc sử dụng scheme đã chỉ định hoặc default

#### Forbid

Được gọi khi người dùng đã xác thực cố tình truy cập vào tài nguyên họ không được phép.

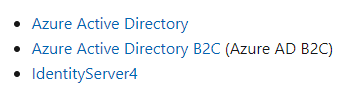
## Sử dụng Identity trên ASP.NET Core

Hỗ trợ các thứ liên quan đến quản lý users, password,...

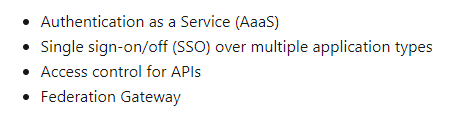
Có thể tạo một tài khoản với login thông tin được lưu trong Identity hoặc có thể sử dụng external login provider (Facebook, Google, Microsoft, Twitter)

Identity được cofing sử dụng SQL Server database để lưu user names, paswords, và profile data

ASP.NET Core identity thêm tính năng UI login vào ASP.NET Core web apps. Để đảm bảo web APIs và SPAs, sử dụng một trong những thứ sau:



IdentityServer4 là một OpenID connect và OAuth 2.0 framework cho ASP.NET Core. Nó cho phép các tính năng bảo vệ như sau



### Authentication và authorization cho SPAs

ASP.NET Core 3.0 hoặc các version về sau đề xuất authentication trong SPAs bằng việc sử dụng support cho API authorization. ASP.NET Core Identity được combined với IdentitySever cho việc thực thi Open ID Connect

Một authentication parameter đã được thêm vào Angular và React project templates. Giá trị cho phép của parameter này là None và Individual.

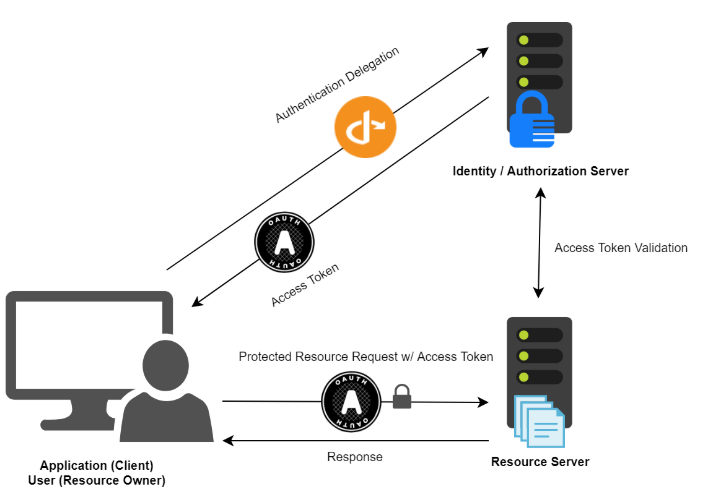
# User Authentication and Identity with Angular, Asp.Net Core and IdentityServer

## Identity and Access Control

Identity như tên của nó, có nghĩa là một số attributes cái mà computer system có thể sử dụng để đại diện a person, organization, application or device. Access control nhắm tới kỹ thuật bảo mật được sử dụng để điều chỉnh người hoặc cái gì đó có thể truy cập vào tài nguyên của họ trong môi trường máy tính

## OpenID Connect và OAuth

OIDC là a simple identity and authentication protocol layer xây dựng trên nền OAuth protocal cái mà cho phép app xác minh end-users.

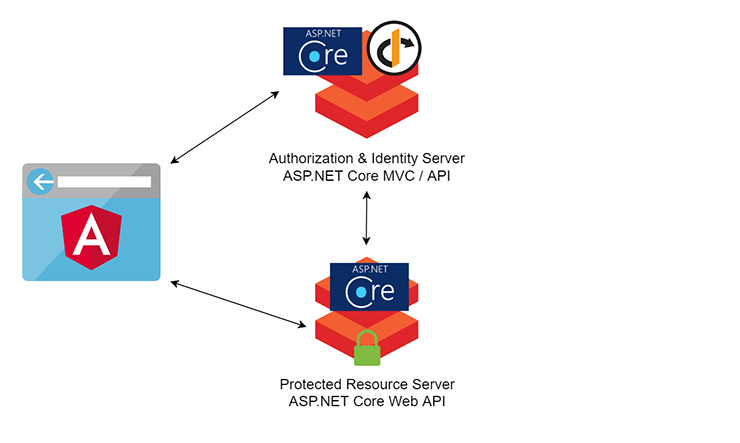


OIDC/OAuth flow

1. User bắt đầu một request với authorization server
2. Nếu các chứng chỉ hợp lệ và tất cả mọi checks thì the authorization server cấp cho the client application một access token
3. the access token được đính kèm tới mỗi request tiếp theo để bảo vệ cho Resource Server
4. Authorization server xác minh access token; nếu thành công thì request cho tài nguyên được bảo vệ thì được cấp và trả về lại client

Giờ ta sẽ áp giao thức trên vào .NET Core và angular

## Architecture



### AuthServer

the AuthServer là OpenID Connect và OAuth provider của chúng ta



Package 1 là OpenID Connect và OAuth framework cho .NET Core

Package 2 là Identity tích hợp với IdentityServer4

## IdentityServer4

### API Access

Application có 2 cách cơ bản để giao tiếp với APIs – sử dụng application identity, hoặc delegating the user’s identity.

OAuth2 là một giao thức cho phép các applications yêu cầu access tokes từ một sercurity token service và sử dụng chúng để giao tiếp với APIs (kiểu như là muốn vào căn cứ quân sự thì cần phải có giấy phép được cấp của bộ quốc phòng, khi đến căn cứ thì bên quân sự sẽ gửi hình ảnh về giấy phép để chứng thực trên bộ quốc phòng xem hết hạn hay không đúng).

# Front-end

Với Front-end, trước tiên sẽ đi sâu vào angular

## Kiến trúc của angular

Angular là một platform và framework mục đích để build single-page applications bằng việc sử dụng HTML và TypeScript

NgModule – là building block, cung cấp context cho components. Angular app là một tập các NgModule. Một app luôn có ít nhất một thằng root module

Component – định nghĩa view, sử dụng các Service được cung cấp bằng cách sử dụng DI

Như vậy có 3 khai niệm cơ bản tạo lên một app là module - component – service. Chúng được định nghĩa bằng các class và sử dụng decorator để đánh dấu và cung cấp metadata nói cho angular biết no là cái gì, giữ nhiệm vụ gì và các thứ liên quan đến nó. Với Module là @NgModule, component là @component, và với service là @service

### Module

Ý nghĩa của NgModule là khai báo một context cho một tập các components chuyên dụng cho một domain, workflow. Hiểu đúng nghĩa thì nó đúng là một module theo đúng gốc, làm một module chuyên về cái gì đó, đảm nhiệm vài trò về lĩnh vực, domain. Trong module thì có các thành phần liên kết, tương tác với nhau để xây dựng thành một module với tính năng hoàn chỉnh là các component. Một module lớn cũng có thể có các module nhỏ

#### Metadata

Một module được định nghĩa bởi class và có decorated là @NgModule. Decorator này là một hàm giữ một object, object này mô tả toàn bộ thành phần của module. Còn bên trong class thì chẳng có gì, các thứ của module đều nằm trong object mà metadata sở hữu

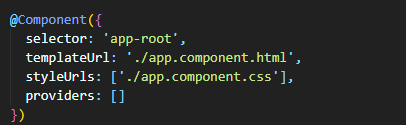
Cấu tạo của object đã đề cập bao gồm các thành phần như sau:

* declarations – khai báo các components, directives và pipes mà module này quản lý
* exports – một tập con của declarations, khai báo những thằng mà phơi ra để các NgModule khác có thể dung
* imports – sử dụng các thành phần mà được export từ module khác mà ta muốn sử dụng cho các template của các component trong module này
* providers – các services mà module này tạo và đóng góp vào khối service toàn ứng dụng
* bootstrap – main view của app, chỉ có root module mới set thằng này, module thành phần không quan tâm

### Component

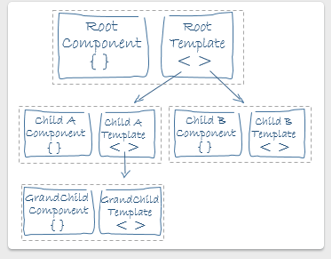
Một component nắm giữ một view.

#### Metadata



* selector – giá trị của thằng này đóng vai trò làm name tag khi sử dụng nó bên trong một view khác
* templateUrl – nơi định nghĩa giao diện của nó
* styleUrls – giá trị là một mảng css phục vụ cho view
* providers – giá trị là một mảng các serives cần thiết cho component

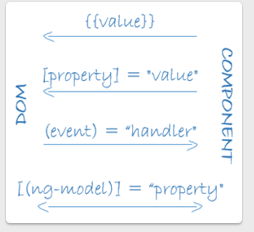
#### Template và views



Như ta có thể nhìn, khi mà khai báo một template trong một view template khác thì nó sẽ có đi kèm luốn cả phần component đi theo luôn và ẩn bên dưới template đó. Root component mà muốn đạt được thằng child component thì phải dựa vào cách.... ở đây chưa đủ kiến thức để biết

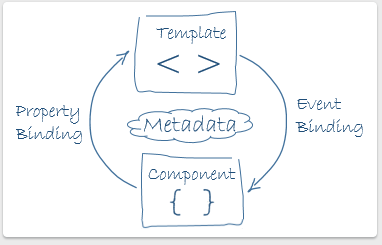
#### Data binding

Như react thì ta phải đẩy data vào trong HTML, với Angular thì nó hỗ trợ kỹ thuật binding 2 chiều

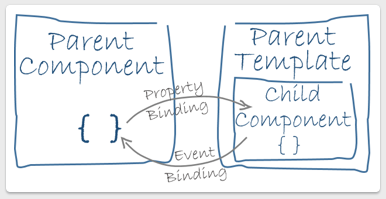


Trên hình là syntax tương ứng với các chiều

* {{value}} – là nội suy display (tạo ra giá trị làm content của html element)
* [property] – là binding vào property của component thuộc template đấy
* (event) – gán event của html element tới hàm xử lý bên dưới component
* [(ng-model)] – có dấu () là từ DOM – component, [] là từ component –DOM, nên có cả 2 là two-way. với thằng này thì thay đổi ở bất cứ đâu thì thằng còn lại cũng thay đổi theo



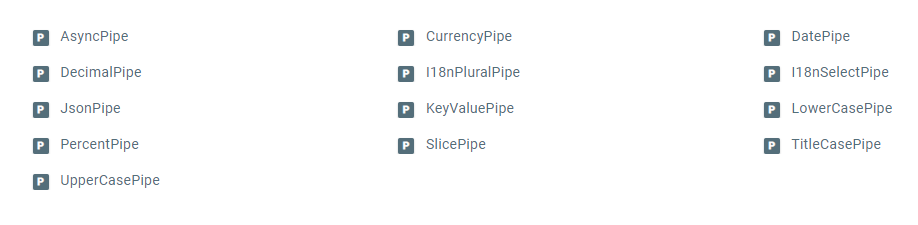
Giao tiếp giữa Parent component với Child component là hoàn toàn thông qua việc binding này. Parent sẽ không thể trực tiếp dùng child component mà chỉ binding data để điều hướng child component (trong WPF ta có Dependency Property để đảm nhiệm vai trò này)



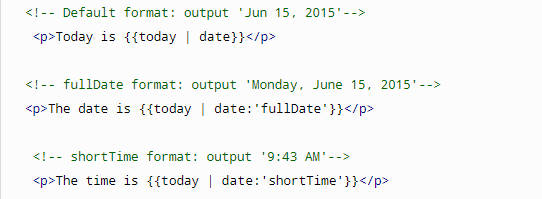
#### Pipes

Thằng này là cho phép chúng ta chuyển đổi định dạng hiển thị ngay trên HTML. Một class mà có @Pipe decorator cần định nghĩa 1 hàm chuyển đối giá trị input tới output cho việc display trên view

Angluar định nghĩa một vài pipes như



Ví dụ về pipe date



#### Directives

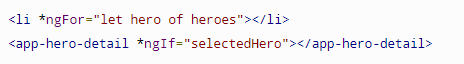
Angular template thì dynamic. Khi angular renders, nó thay đổi DOM theo các directives. Một Directive là một class có @Directive decorator

Một component về kỹ thuật thì chính là một directive (vì nó phải render theo component đó định nghĩa mà). Tuy nhiên, các component rất đặc biệt và cũng là trung tâm của Angular app vì vậy nó được lấy là @Component, extend @Directive

Có 2 loại directive là structural và attribute, angular định nghĩa một số của 2 thằng này, và ta cũng có thể tự định nghĩa nếu cần

##### Structural directive

Gọi là structural vì nó thay đổi kết cấu của DOM bằng cách thêm, xóa, thay thế các element trong DOM. Ví dụ



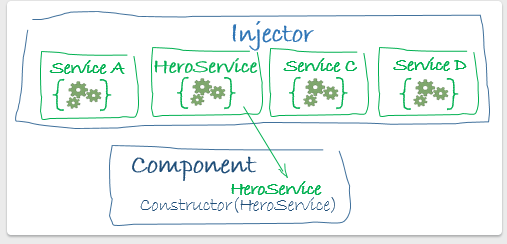
##### Attribute directive

Thay đổi sự xuất hiện hoặc hành vi của element đã có

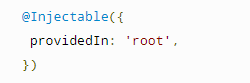
### Service và DI

Service là các tính năng, các thành phần sẽ quan tâm tới các tính năng phần bên dưới, hỗ trợ cho các component, để mà các component có thể sử dụng được các service hỗ trợ này thì các service phải được chứa trong DI.

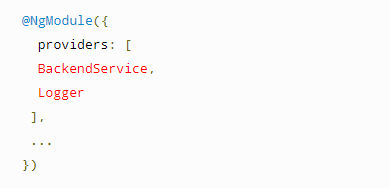
Định nghĩa một service thì đơn giản là @Injectable và theo sau là class định nghĩa serivce đó



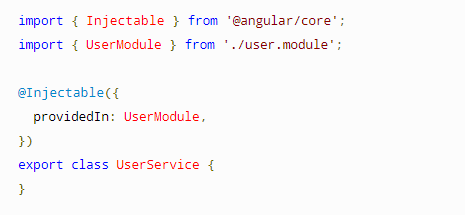
Đăng ký service vào trong các provider khác nhau. Có 3 cấp độ provider là root, module và component. Mặc định là root



Với module thì ta sẽ đăng lý nó trong metadata của module và khi đó ta có thể sử dụng nó ở toàn bộ các component trực thuộc module đó



hoặc một cách ưa thích hơn là khai báo như sau



## Syntax template

### Interpolation and Template Expressions

{{...}} – gọi là Interpolation, thằng này nhúng expressions (biểu thức) vào trong text. Ví dụ



text giữa các dấu ngoặc nhọn thường là tên property trong component. Angular thay thế tên này với giá trị tương ứng của nó trong component

Nói chung là {{...}} dùng để nhúng expression vào trong bất cứ chuỗi string nào, phần ... sẽ được angular tính toán và sau cùng mới convert sang string. ví dụ



#### Template Expression

Là thằng mà sẽ được đưa vào giữa các dấu ngoặc nhọn của Interpolation. Angular execute expression và gán giá trị của nó tới một property binding target; target ở đây có thể là HTML element, component, directive

{{1+1}} sẽ có template expression là 1+1. Trong dạng property binding (ví dụ ở đây đang là dạng nhúng) thì expression sẽ ở bên phải dấu bằng [property]=”expression”

Vậy expression là thằng nó nằm trong tương tác của

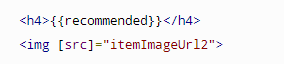
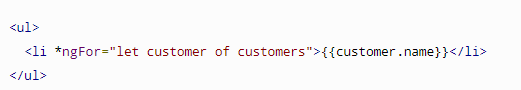
{{expression}} – nhúng

[property]=”expression” – property binding

#### Expression context

Có vài loại context ở expression refer đến

Khá là giống với cơ chế binding trong WPF

* Component context – tức là nó refer đến các property có trong component
* Template context – refer đến property ngay trên template

Context trong giới hạn của expression là sự trộn lại của template variables, directive’s context object và component’s members. Nếu mà có một tên bị trùng của nhiều context thì ưu tiên đầu là template variable, sau đó là tên trong directive và cuối cùng là component

#### Template statements

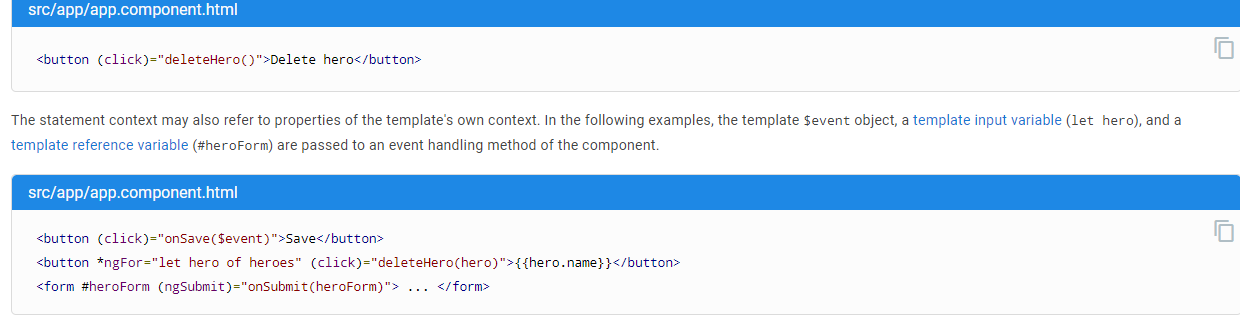
expression là cho property binding và embeded, còn thằng chịu trách nhiệm cho event (expression là tính toán rồi pass sang string còn statement là khai báo ra rồi để đấy)

(event)=”statement”



#### Statement context

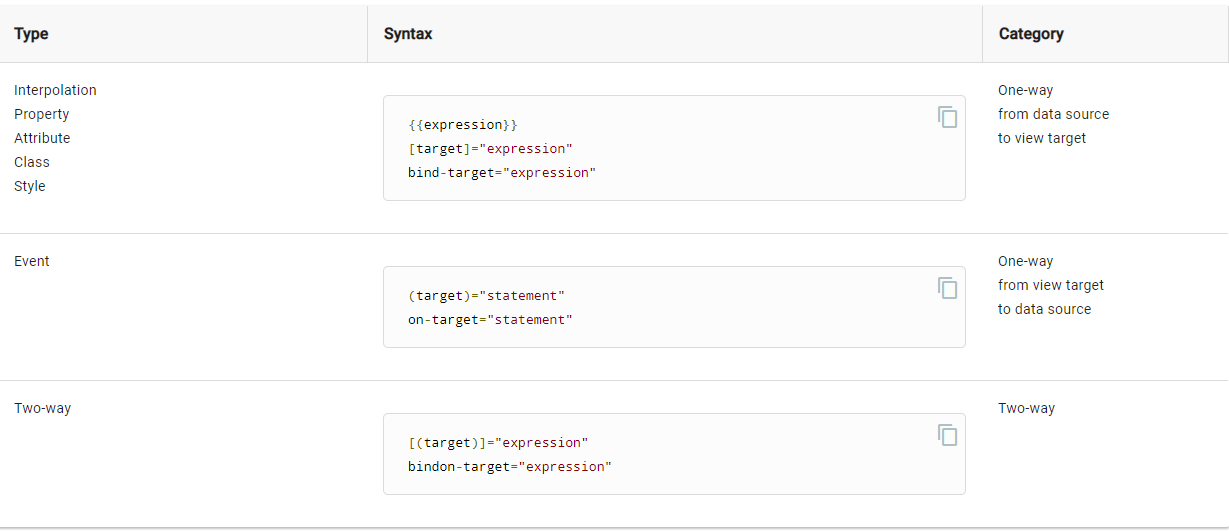
Nó chỉ có context ở component, nó cũng có thể refer tới các properties của template



ưu tiên sẽ là: component > template

Có 2 loại variable của template là input và reference variable. input là thằng trong ngFor còn reference là #heroForm

#### Binding



#### HTML attributes vs DOM property

Attributes được định nghĩa bởi HTML. Properties được truy cập từ DOM nodes

Template binding làm việc với properties và events chứ không phải attributes

<input type="text" value="Sarah">

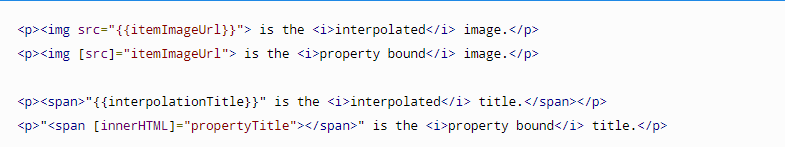
Khi browser render đoạn trên, nó tạo một DOM node với value property được khởi tạo là “Sarah”

Khi user nhập vào “Sally” thì DOM element value property trở thành “Sally”. Tuy nhiên, nếu ta nhìn vào HTML attribute value bằng việc sử dụng input.getAttrribute(‘value’) thì ta thấy giá trị của nó là ‘Sarah’

* attribute vlaue chỉ làm công tác initial value; DOM value property là current value

Property binding và interpolation

Với interpolation thì luôn trả ra một giá trị là string, còn property binding nó sẽ parse dữ liệu theo đúng type của property . Nên interpolation thì có thể thay thế khi type của property là string

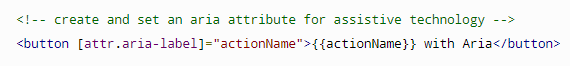


#### Attribute, class, and style bindings

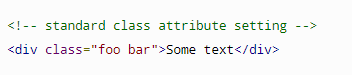
##### Attribute

Đây chỉ là ngoại lệ duy nhất, thường thì property binding thì thích hợp hơn. Tuy nhiên, thi thoảng chúng ta đếch có property đê binding nên giải pháp sẽ là binding và attribute

Syntax thì thống với property binding nhưng thay vì bên trong [] là property của element thì nó bắt đầu với attr, theo sau là dot (.) và tên của attribute

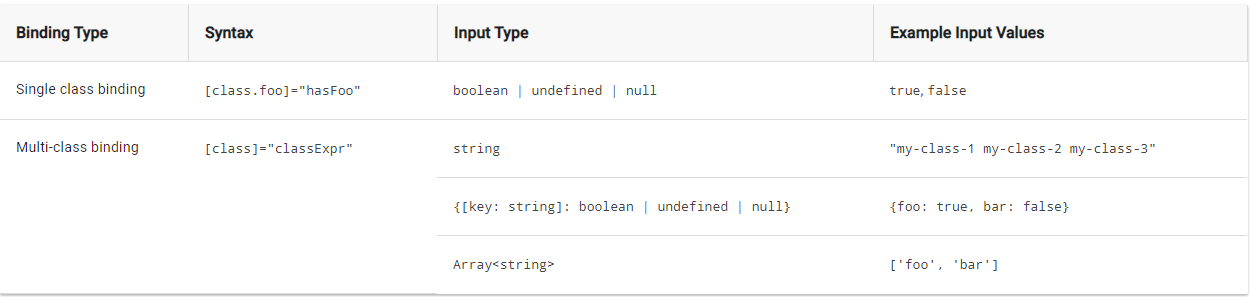


##### Class

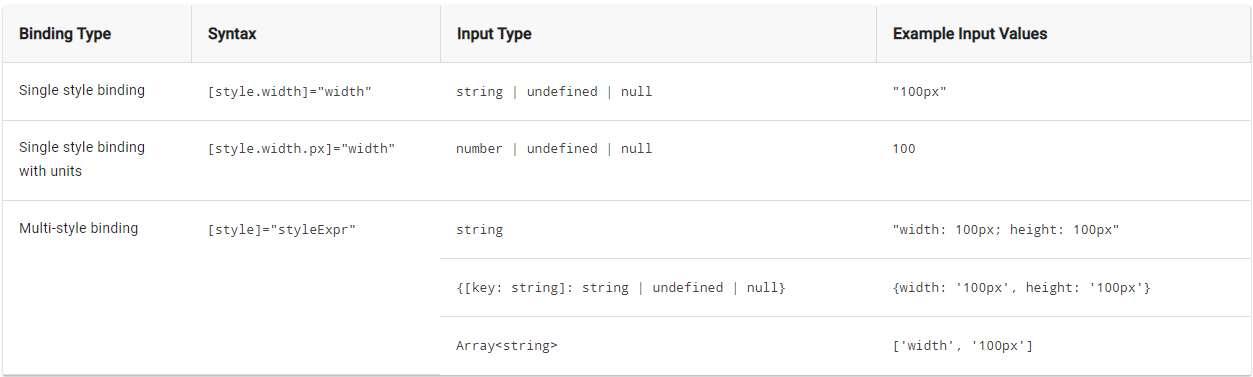


Code bên trên là khi ta setting class một cách bình thường với các giá trị tĩnh

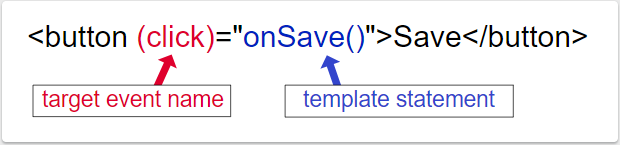
Ta có thể add và remove CSS class name từ element’s class attribute với class binding



##### Style



#### Event binding



the target là button’s click event

<button (click)="onSave($event)">Save</button>

Có thể thay thế bằng

<button on-click="onSave($event)">on-click Save</button>

##### $event và event handling statements

Trong một event binding, Angular sét up một event handler cho target event

Khi mà event được raised, handler execute template statement.

$event là thông tin được bao gồm, target event sẽ chỉ định shape của $event object.

##### Custom events với Event Emitter

giống delegate, nó chính là property để có thể kích hoạt gọi một hàm nào đó từ bên ngoài truyền vào trong nội bộ của child element khi mà parent element truyền tới

#### Two-way binding [(...)]

### Built-in directives

Có 2 built-in directives là attribute directive và structural directive

#### Attribute directive

##### NgClass

Add hoặc remove một vài CSS classes (nếu sử dụng HTML thường ta muốn làm cái này phải qua jquery)

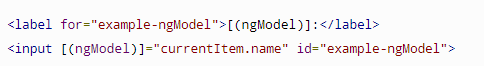
<div [[ngClass](https://angular.io/api/common/NgClass)]="isSpecial ? 'special' : ''">This div is special</div>

##### NgStyle

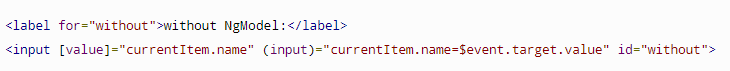
Cả NgStyle và NgClass phục vụ cho việc add thêm hoặc remove, cùng tồn tại với các class đã có từ trước. Còn attribute class binding là set ghi đè cho tất cả

##### [(ngModel)]

Cho phép hiển thị data property và update property đó khi người dùng make change



Thằng này tương đương với



Bản chất thằng input không có property nào vừa là để hiển thị vừa nghe luôn event input (trong WPF thì được tích hợp luôn vào trong property Text)

#### Structural

##### NgIf

\*ngIf=”condition expression”

If và hidden - 2 thằng này khác nhau hoàn toàn, mặc dù cho ra hiện tượng giống nhua trên giao diện. với If thì nó xóa thẳng component đó trong DOM, còn hidden thì vẫn còn, chỉ là nó ẩn đi

Ngoài ra ngIf còn có thể check null, nếu null thì không render

<div \*[ngIf](https://angular.io/api/common/NgIf)="nullCustomer">Hello, <span>{{nullCustomer}}</span></div>

##### NgFor

Ta định nghĩa một block của HTML cái là một single item được displayed và sau đó nói với Angular là sử dụng block này là một template cho việc render mỗi item trong một list. \*ngFor chính là cách để block nói với Angular

<div \*[ngFor](https://angular.io/api/common/NgForOf)="let item of items">{{item.name}}</div>

Đoạn string gán cho \*ngFor không phải là template expression, nó mà microsyntax do Angular cũng cấp

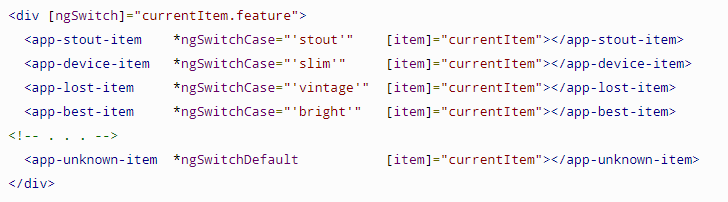
Với cú pháp có index

<div \*[ngFor](https://angular.io/api/common/NgForOf)="let item of items; let i=index">{{i + 1}} - {{item.name}}</div>

track – nâng cao hiệu năng khi list quá lớn, mặc định với list nhỏ thì có lẽ không cần dùng

##### NgSwitch

Nó là một tập của 3 directive gồm NgSwitch, NgSwitchCase, NgSwitchDefault



##### Template reference variable (#var)

một template reference variable thường là một reference tới một DOM element bên trong một template.

Sử dụng hash symbol (#) để khai báo reference variable

<input #phone placeholder="phone number" />

Cái này cũng giống như WPF với đặt tên cho control và gọi nó trên control khác

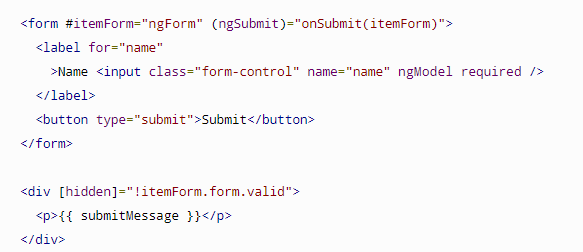
<input #phone placeholder="phone number" />

<!-- lots of other elements -->

<!-- phone refers to the input element; pass its `value` to an event handler -->

<button (click)="callPhone(phone.value)">Call</button>

Ngoài việc set reference cho một element đã khai báo, ta có thể set nó cho NgForm directive



Cái này sẽ cần tìm hiểu sâu hơn khi dùng đến

### @Input() và @Output() properties

2 thằng này đóng vai trò là phương tiện để giao tiếp giữa parent component và child component

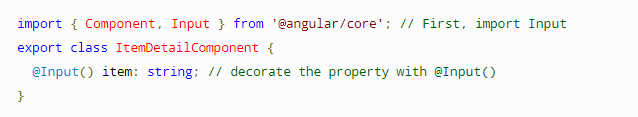
@Input sẽ là cha gửi thông điệp tới con

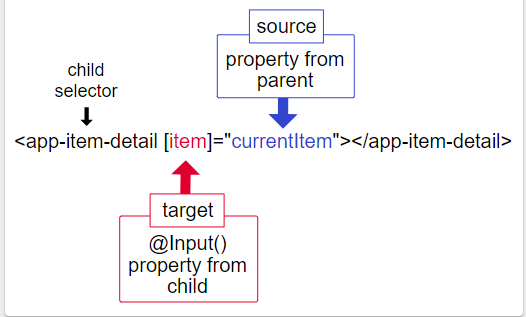
@Output là con gửi thông điệp tới cha ( bằng cách là con giữ một hàm trong thằng cha, khi con thông báo sẽ kích hoạt để hàm đó hoạt động ngay trong thằng cha)



ở trên, <child-component> selector hay child directive, được nhúng vào trong một <parent-component>

Input và Output ở đây là đang nói đến việc đối xử với child component, input là đầu vào cho child component, ouput là child component nhả ra, trả lại





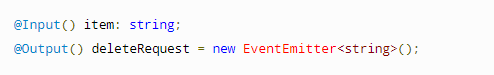
Với Output thì như sau



Cái này nó khá tương tự như Func<object,string> nhưng vì javascript và typescript không cần thiết có kiểu trả về nên ở đây chỉ khai báo kiểu dữ liệu input mà thôi. Thằng này đơn giản là khai báo một property với tên là newItemEvent thuộc kiểu EventEmitter<string> tức là chỉ cho phép gán các hàm có 1 parameter với kiểu string



Chốt lại, khai báo như sau



Nếu giả dụ ta muốn sử dụng 2 tiên , một tên sử dụng trong nội bộ component và một tên sử dụng để bên ngoài câu vào thì khai báo như sau

