# PIPE

## Pipe là gì?

* Pipe được dùng để chuyển đổi dữ liệu mà ta hiển thị lên template cho người dùng có thể hiểu được. Hiểu đơn giản, pipe là một function nhận **input** mà chúng ta truyền vào, và **output** ra giá trị mình mong muốn.
* Ví dụ là định dạng kiểu hiển thị datetime, viết hoa chữ cái, hiển thị tên thành phố, định dạng lại số hay đơn vị tiền, ...
* Điểm lợi thế của Pipe là dễ tái sử dụng. Vì thông thường sẽ có khá nhiều page cần hiển thị date time, việc dùng Pipe sẽ đem lại nhiều ưu điểm hơn là function.

## 1.2 Cú pháp

|  |
| --- |
| {{data | pipeName : (Parameter1) : (Parameter2) ...}} |

* Trong đó:
* **data**: là dữ liệu cần chuyển đổi
* **pipeName**: là pipe mà bạn sử dụng
* **Parameter1, Parameter2**: là các tham số truyền vào cho pipe
* Pipes cho phép chúng ta dùng nhiều pipe để transform một value, cú pháp có dạng như sau:

|  |
| --- |
| {{ interpolated\_value | pipe\_name\_1 | pipe\_name\_2 |... | pipe\_name\_n }} |

Sau khi có output **pipe\_name\_1**, thì output này sẽ được xử lý qua **pipe\_name\_2** rồi tương tự đến **pipe\_name\_n** để ra output cuối cùng hiển thị lên UI

## 1.4 Sử dụng Pipe

### *1.4.1 Các pipe có sẵn đi kèm với Angular (built-in pipe)*

Angular cung cấp sẵn khá nhiều pipes để có thể sử dụng được khi import CommonModule từ package @angular/common.

|  |  |
| --- | --- |
| **Pipe** | **Mô tả** |
| [DatePipe](https://angular.io/api/common/DatePipe) | Formats a date. |
| [UpperCasePipe](https://angular.io/api/common/UpperCasePipe) | Convert text sang chữ hoa. |
| [LowerCasePipe](https://angular.io/api/common/LowerCasePipe) | Convert text sang chữ thường. |
| [CurrencyPipe](https://angular.io/api/common/CurrencyPipe) | Hiển thị giá trị tiền tệ. |
| [DecimalPipe](https://github.com/angular-vietnam/100-days-of-angular/blob/master/https:/angular.io/api/common/DecimalPipe) | Hiển thị số thập phân |
| [PercentPipe](https://angular.io/api/common/PercentPipe) | Hiển thị phần trăm % |
| [JsonPipe](https://angular.io/api/common/JsonPipe) | Hiển thị json |
| [AsyncPipe](https://angular.io/api/common/AsyncPipe) | Hiển thị value của observable và tự động unsubscribe khi view được destroy |

Toàn bộ các pipe có sẵn trong CommonModule: *https://angular.io/api?type=pipe*

### *1.4.2 Viết custom pipe*

Trong quá trình xử lý dữ liệu để hiển thị ra cho người dùng thì các pipe mà Angular hỗ trợ sẵn thì sẽ là chưa đủ so với yêu cầu mà bài toán đưa ra, để có để đáp ứng được yêu cầu đó thì ta có thể tự viết ra các pipe để phù hợp với bài toán.

|  |
| --- |
| **ng g pipe my-pipe** |

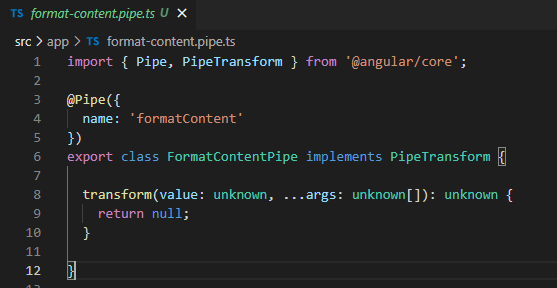
* ***Cú pháp:***
* ***Demo:***

Tạo ra 1 pipe cú nhiệm vụ là cắt string khi chữ nó quá dài thì sẽ hiển thị ...



* Sau khi chạy lệnh tạo pipe ở trên xong thì ta sẽ có 1 file như sau:

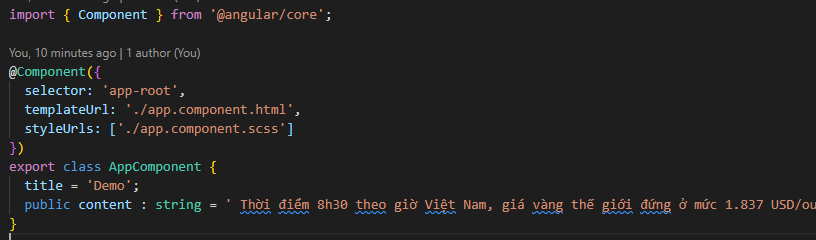
**format-content.pipe.ts**



***Trong đó :***

* **value**: sẽ là dữ liệu đầu vào của pipe
* **args:** sẽ là các tham số truyền vào cho pipe
* Tiếp theo ta sẽ custom lại cái pipe để cho nó hiển thị được như yêu cầu trên.

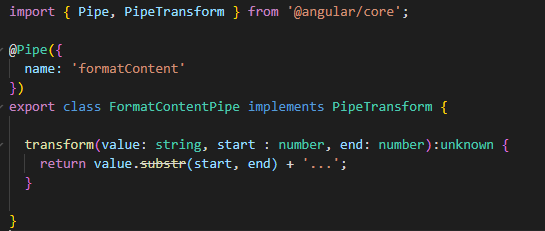
**app.component.ts**



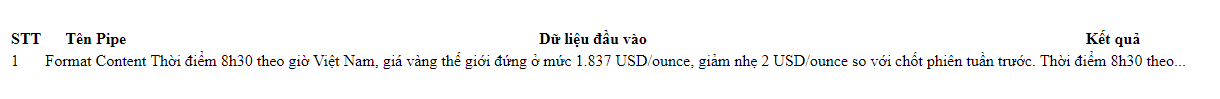
**app.component.html**



**format-content.pipe.ts**



Ở pipe này mình nhận dữ liệu đầu vào kèm theo 2 tham số được truyền vào cho pipe sau đó mình xử lý dữ liệu với hàm **substr()** và kết quả mình thu được là



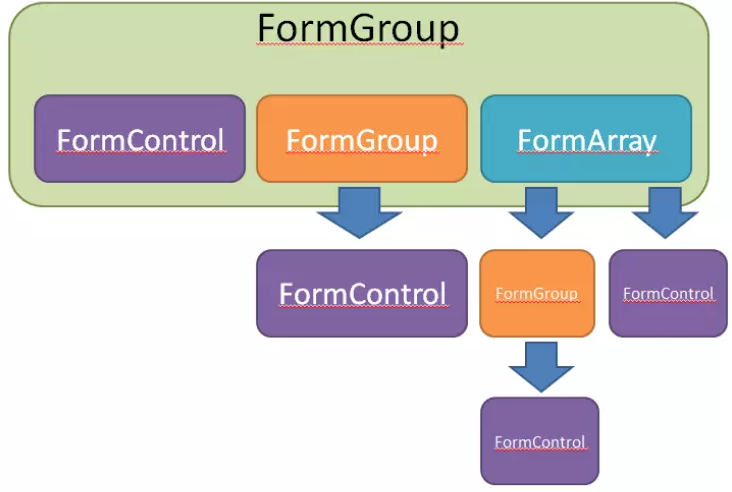
# ANGULAR FORM

## 2.1 Template-driven form

* Phương pháp mà chúng ta sẽ tạo forms dựa vào template. Chúng ta thực hiện việc thêm các directives và hành vi vào template, sau đó Angular sẽ tự động tạo forms để quản lý và sử dụng.
* Các directives mà Angular cung cấp có thể kể đến như: ngForm, ngModel, ngModelGroup...

## 2.2 Reactive form (Model Driven Form)

* Thuật ngữ Reactive Forms hay còn được gọi là Model-Driven Forms, là một phương pháp để tạo form trong Angular, phương pháp này tránh việc sử dụng các directive ví dụ như ngModel, required,… thay vào đó tạo các Object Model ở trong các Component, rồi tạo ra form từ chúng. Một điều lưu ý đó là Template-Driven là **async** còn Reactive là **sync.**
* Trong Reactive forms, chúng ta tạo toàn bộ form control tree ở trong code (khởi tạo ngay, khởi tạo trong constructor, hoặc khởi tạo trong ngOnInit), nên có thể dễ dàng truy cập các phần tử của form ngay tức thì.
* Về control tree hay còn gọi là AbstractControl, nó cung cấp các hành vi, thuộc tính chung cho các lớp con, trong Reactive form có 3 lớp con là: FormControl, FormGroup, và FormArray:

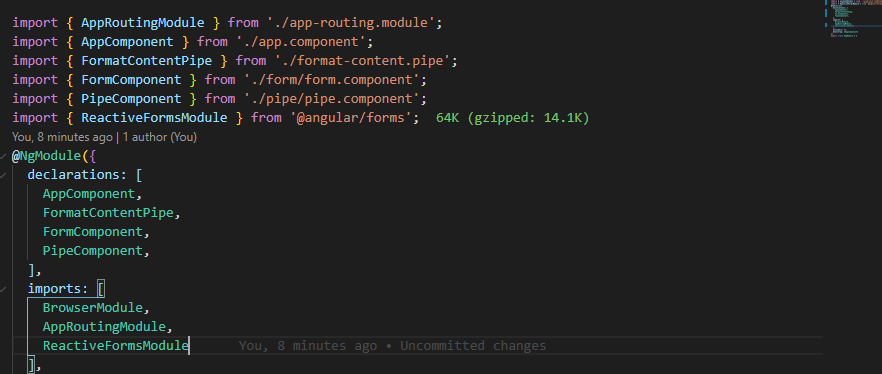


* **FormControl** là lớp thấp nhất, nó lưu giữ giá trị và trạng thái hợp lệ của một form control, nó tương ứng với một HTML form control như input, select, text area…
* **FormGroup** là nơi lưu giữ giá trị và trạng thái hợp lệ của một nhóm các đối tượng thuộc AbstractControl – có thể là FormControl, FormGroup, hay FormArray – đây là một dạng composite. Ở level cao nhất của một form trong component của bạn là một FormGroup.
* **FormArray** là nơi lưu giữ giá trị và trạng thái hợp lệ của một mảng các đối tượng thuộc AbstractControl giống như FormGroup. Nó cũng là một dạng composite. Nhưng nó không phải là thành phần ở level cao nhất.

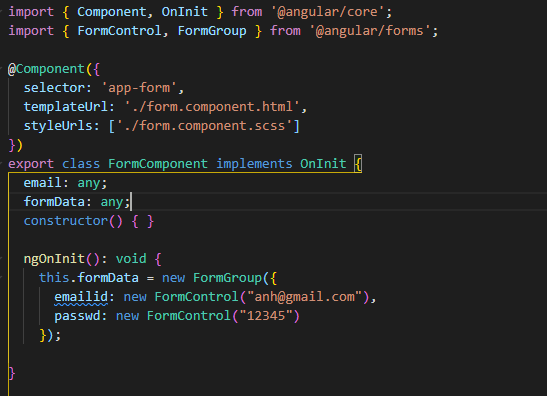
**2.3 Demo**

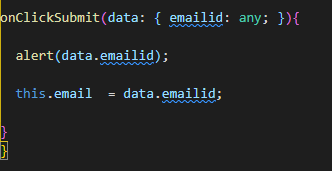
 Trong model driven form để sử dụng được chúng ta cần **import ReactiveFormsModule** từ @angular/forms

App.module.ts

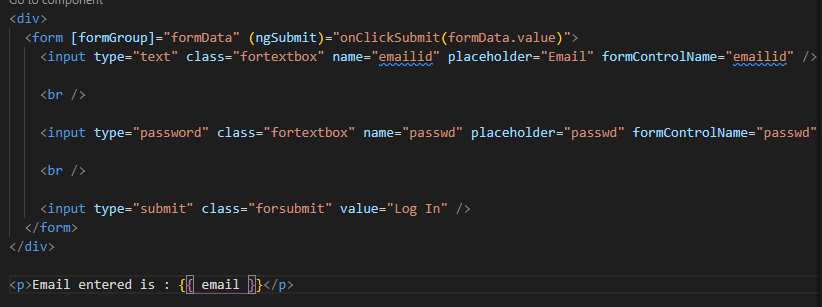


form.component.ts





form.component.html



### Thêm validation tới Form

chúng ta cần nhập Validators từ @angular/ form như hình dưới đây



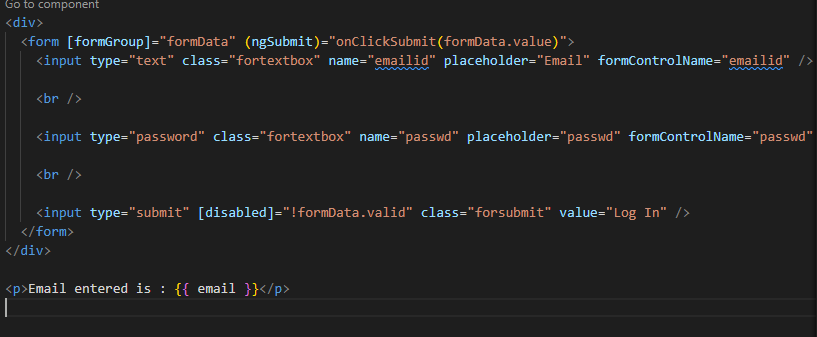
Angular có các trình xác thực tích hợp như trường bắt buộc, độ dài tối thiểu, độ dài tối đa và mẫu. Chúng phải được truy cập bằng module Validators. Bạn chỉ có thể thêm trình xác thực hoặc một mảng trình xác thực cần thiết để cho Angular biết nếu một trường cụ thể là bắt buộc. Bây giờ chúng ta hãy thử tương tự trên một trong các hộp văn bản đầu vào, tức là id email.

Đối với id email, ta đã thêm các thông số xác thực sau

Trong file form.component.ts, thực hiện validator trong angular của form



Để thực hiện được validator form của angular bạn cần phải chỉnh sửa thêm một chút để người dùng có thể biết được các giá trị mà họ nhập sai hoặc thiếu ở html



 [disabled] ="!formdata.valid"  Kiểm tra các giá trị nhập sai

# RxJS (Reactive Extensions Library for JavaScript)

RxJS là 1 thư viện rất tiện lợi cho việc xư lý bất đồng bộ (bao gồm cả việc xư lý dựa trên sự kiện như 1 cú click chuột)

## 3.2 RxJS Concepts

### Observable

* Đại diện cho ý tưởng về một tập hợp các giá trị hoặc các sự kiện trong tương lai. Khi các giá trị hoặc sự kiện phát sinh trong tương lai, Observable sẽ điều phối nó đến Observer.
* Observable chỉ là một function (class) mà nó có một số yêu cầu đặc biệt. Nó nhận đầu vào là một Function, mà Function này nhận đầu vào là một Observer và trả về một function để có thể thực hiện việc cancel quá trình xử lý
  + - Khi chúng ta làm việc với Observable thì sẽ thường xuyên sử dụng phương thức **subscribe** để theo dõi các thay đổi của dữ liệu hoặc nhận về kết quả của một công việc nào đó.
    - Khi API trả về kết quả thì dữ liệu đó sẽ được log ra qua phương thức **subscribe.** Và để kết thúc quá trình lắng nghe kết quả từ API trả về thì chúng ta sẽ xử dụng đến phương thức **unsubscribe.**

**Observer**

Là một tập hợp các callbacks tương ứng cho việc lắng nghe các giá trị (next, error, hay complete) được gửi đến bởi Observable.

### Subscription

Subscription là kết quả có được sau khi thực hiện một Observable, nó thường dùng cho việc hủy việc tiếp tục xử lý.

### Operators

Operators là các pure functions cho phép lập trình functional với Observable.

* Creation Operators.
* Transformation Operators.
* Filtering Operators.
* Combination Operators.
* Error Handling Operators.
* Utility Operators.
* Multicasting Operators.
* Conditional and Boolean Operators.
* Mathematical and Aggregate Operators.

### Subject

Subject để thực hiện việc gửi dữ liệu đến nhiều Observers (multicasting).

### Schedulers

Một scheduler sẽ điều khiển khi nào một subscription bắt đầu thực thi, và khi nào sẽ gửi tín hiệu đi.