Report

Angular tutorial for beginners  
(14/07/2023)

# Service/Dependency Injection:

Chúng ta có thể gặp một tình huống mà chúng ta cần một code để sử dụng ở mọi nơi trên trang. Chẳng hạn như kết nối dữ liệu cần được chia sẻ giữa các component. Với Service, chúng ta có thể truy cập các phương thức và thuộc tính trên các component khác trong toàn bộ dự án.

## Tạo:

ng g s service\_name

## Khai báo:

* Để sử dụng service cho 1 module, khai báo nó trong providers

## Dependency Injection:

* Là 1 Design pattern
* Dependency: Phụ thuộc, Injection: Tiêm (đưa 1 cái gì đó cực ì nhỏ vào đối tượng lớn hơn)
* Dependency Injection: nhằm đứa service và component bất kì để nó có thể sử dụng service đó thông qua một đối tượng thuộc lớp service được khai báo

# Observables:

## Tạo file json để fake API

* Install json-server: npm i -g json-server
* Tạo db.json trong approot
* Xem thông tin db: json-server --w db.json

## Hoạt động:

Hành vi observable tuân theo mô hình lập trình của một trình observer có hai yếu tố chính đó là observable và observer. Các observable gửi dữ liệu trong khi observer đặt mua nó để nhận được dữ liệu. Các observable bắn dữ liệu để đáp ứng với một sự kiện. Ví dụ: khi người dùng nhấp vào nút click để phản hồi dữ liệu nhận được từ máy chủ từ xa.Mặt khác, Observer có ba cách xử lý để sử dụng dữ liệu mà nó nhận được:

* onNext: xử lý dữ liệu được yêu cầu
* onError: để xử lý lỗi
* onComplete: được sử dụng khi quá trình kết thúc

## Cú pháp:

* Service:

getPokemon() : Observable<Pokemon[]> {

    return this.http.get<Pokemon[]>(POKEMON\_API);

  }

* Component:

.getPokemon.subscribe((data : Pokemon)=>{

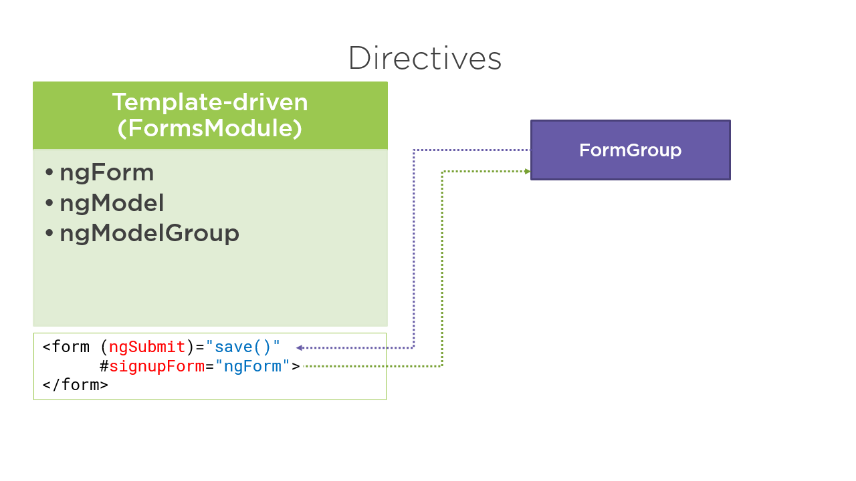
      this.pokemon = data;})

# Template Driven Forms:

## Khái niệm:

Template-driven form là phương pháp mà chúng ta sẽ tạo forms dựa vào template (giống như trong Angularjs). 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.

## Cấu trúc cơ bản:



<form #itemform="ngForm"> //Template variable #itemform

    <input name="NamePK" type="text" [(ngModel)]="pokemon.namePK" />

//Attribute name name=“NamePK”

    <input name="TypePK" type="text" [(ngModel)]="pokemon.typePK" />

</form>

Form: {{ itemform.value | json }}

// Định dạng của itemform.value | json

{

“NamePK”: “”,

“TypePK”: “”

}

## Activity flow:

* User thay đổi dữ liệu trong các ô input như là input, checkbox, text area...
* Input Element sẽ gửi đi input event chứa value từ user vừa nhập vào
* Control\_Value\_Accessor sẽ trigger phương thức setValue( ) trên FormControl instance.
* FormControl instance sẽ emit giá trị mới qua valueChanges observable
* Bất cứ subcriber đến valueChanges observable đểu nhận được giá trị mới nhất của control đó.
* Control\_Value\_Accessor cũng gọi phương thức NgModel.viewToModelUpdate( ) để emit ngModelChange event.
* Trong trường hợp chúng ta sử dụng cơ chế two-way binding thì lúc này property trong component sẽ được update value bởi ngModelChange event).
* Khi property trong Component thay đổi thì cơ chế phát hiện thay đổi bắt đầu chạy và control\_Value\_Accessor sẽ update Input Element trong view với giá trị mới nhất của property trong component.

## Một vài thành phần con:

### Radio box:

<input type="radio"

       name="Favorite"

       [value]="false"

       [ngModel]="pokemon.favorite"

      (ngModelChange)="toggleIsFav($event)" //event trả về [value]

/>

### Check box:

<input type="checkbox"

       name="InBag"

[(ngModel)]="pokemon.inBag"

(ngModelChange)="onChoose($event)"

/>

### Select/Option:

<select (ngModelChange)="onSelect($event)" name="pokemonType" [(ngModel)]="pokemon.typePK">

        <option \*ngFor="let type of pokemonType"

                [value]="type.value"

                [selected] = "type.value===pokemon.typePK">

                {{ type.value }}

        </option>

      </select>

## Form Validation: