Raising the bar

Angular Form

Mục tiêu

- Sử dụng được cơ chế Data Binding với form.
- Validate được dữ liệu nhập vào.
- Sử dụng được các loại input khác nhau.
- Xử lý được các input event

Template-driven Form

ngForm và ngModel directives

- ngForm directive sẽ được sử dụng để khởi tạo form.
- ngModel directive được sử dụng để đăng ký một control với ngForm, nó yêu cầu attribute **name** để đăng ký.
- [ngModel]="property" để binding data cho input.
- [(ngModel)]="property" để sử dụng two-way binding.

ngForm và ngModel directives

```
<form  #profileForm="ngForm"
   (submit)="onSubmit(profileForm)"
   novalidate>
</form>
```

Khai báo template variable tên là profileForm, nó lưu trữ một object là instance của ngForm directive

Lắng nghe event khi form thực hiện submit, sau đó method onSubmit sẽ được gọi, với tham số đầu vào là template variable chúng ta đã tạo

ngForm và ngModel directives

```
<input
    ngModel
    name="username"
    class="form-control"
    type="text" id="example-text-input">
```

ngModel directive kết hợp với attribute "name" để đăng ký control này với ngForm

ngForm và ngModel directives

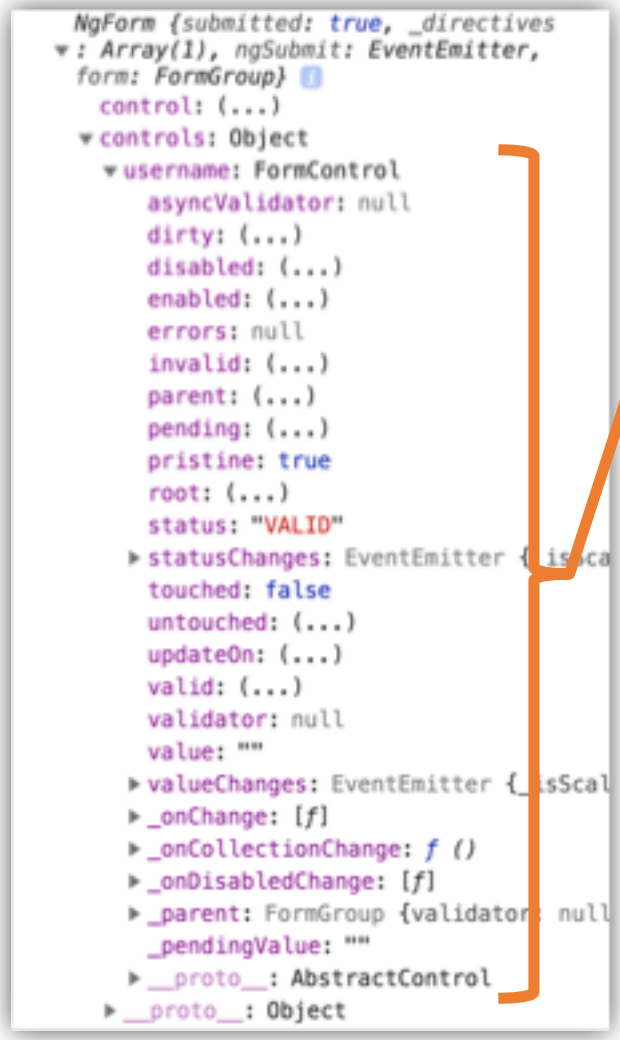
```
class TemplateDrivenFormComponent {

  onSubmit(profileForm) {
    console.log(profileForm);
  }

}
```

Method này sẽ được thực thi khi submit form

ngForm và ngModel directives



Đây là cấu trúc của class để mô tả một form control

ngModel vs [ngModel] vs [(ngModel)]

```
class TemplateDrivenFormComponent {
  profile = {
    username: 'Bob',
    email: 'abc@deg.com',
    facebook: 'facebook.com',
    twitter: 'twitter.com',
    website: 'example.com',
    tel: '1234-5678-90'
  };
}
```

Dữ liệu cần binding cho các control của form

ngModel vs [ngModel] vs [(ngModel)]

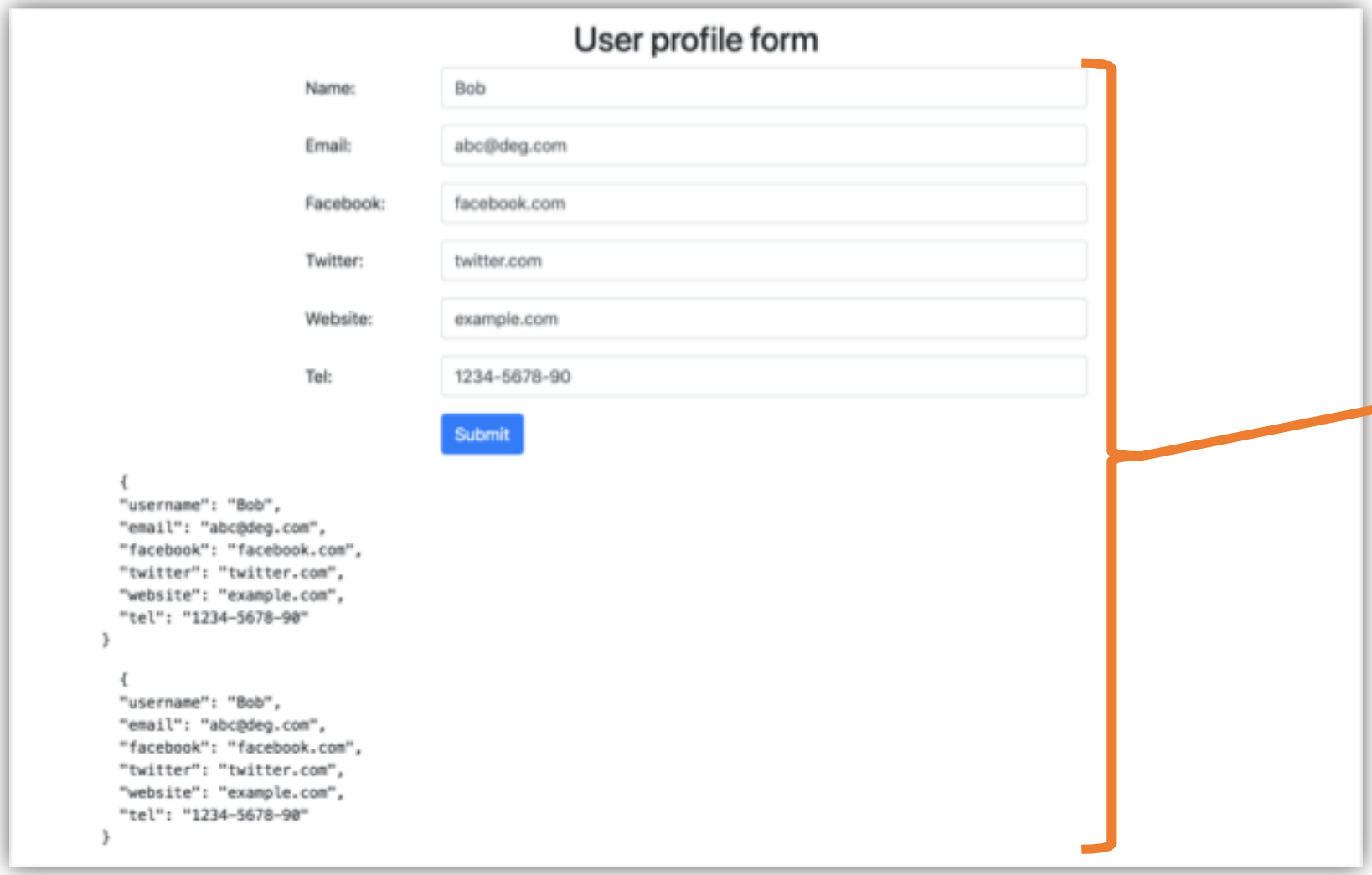
```
<input
   [ ngModel] =" profile.username"
   name="username"
   class="form-control"
   type="text"  id="example-text-input">
```

ngModel vs [ngModel] vs [(ngModel)]

```
<pre>
  {{profileForm.value | json}}
</pre>
<pre>
  {{profile | json}}
</pre>
```

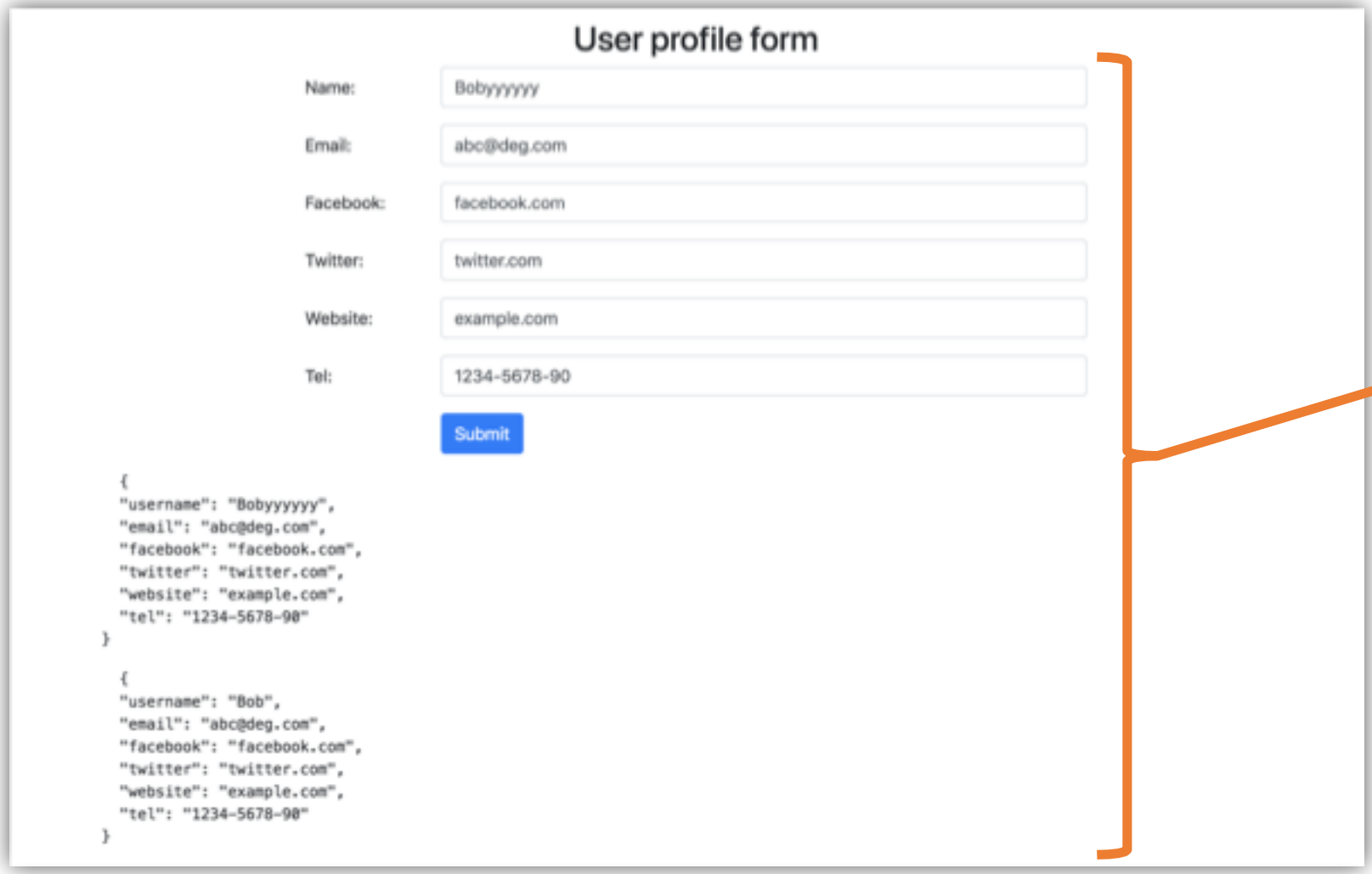
Compare giá trị của object đang chứa và giá trị mà form đang chứa

ngModel vs [ngModel] vs [(ngModel)]



Dữ liệu đã được binding

ngModel vs [ngModel] vs [(ngModel)]



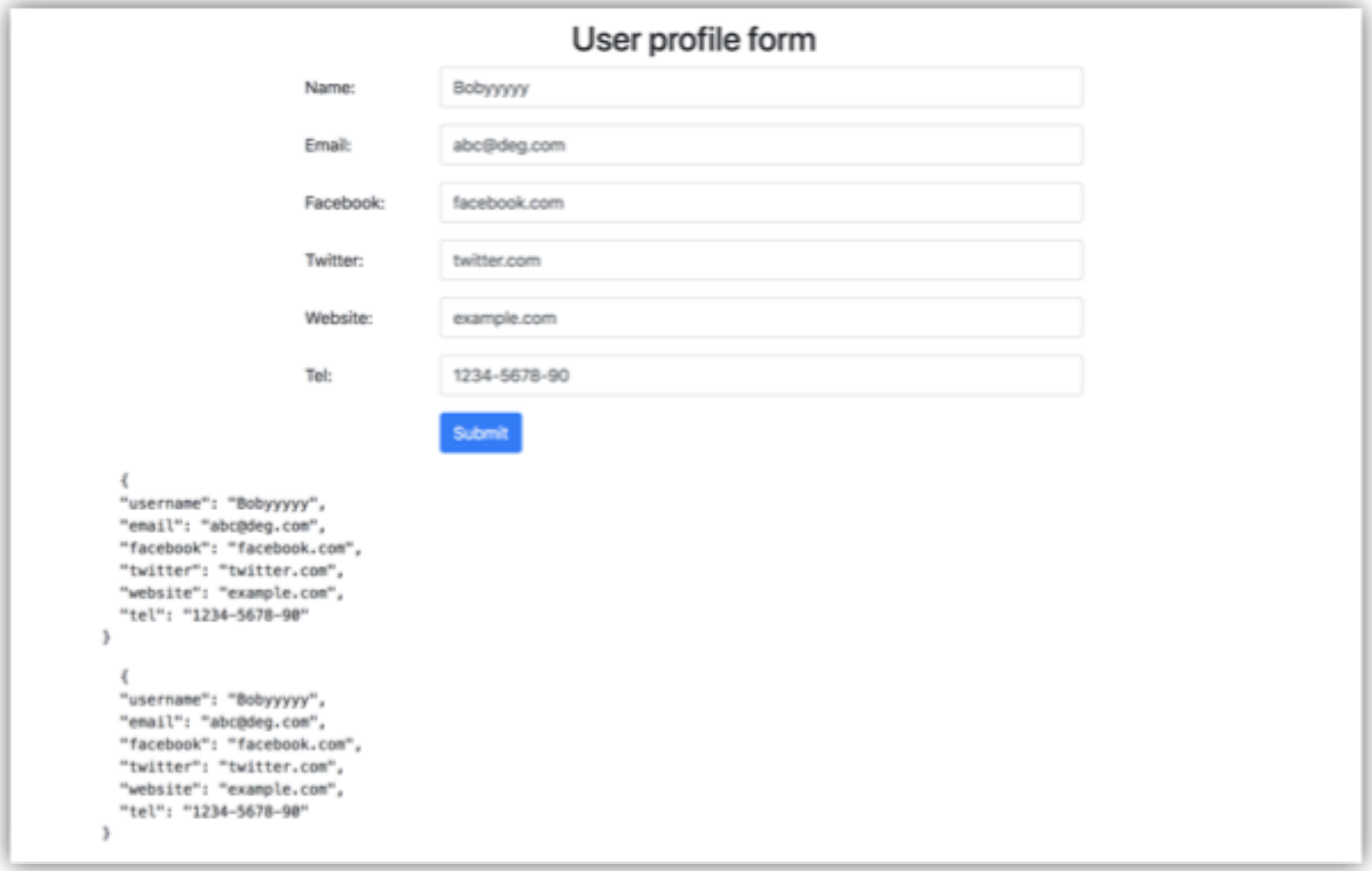
Mặc dù dữ liệu của form đã update, nhưng dữ liệu của model không được update

ngModel vs [ngModel] vs [(ngModel)]

```
<input
   [(ngModel)]="profile.username"
   name="username"
   class="form-control"
   type="text" id="example-text-input">
```

Two-way binding cho control và model

ngModel vs [ngModel] vs [(ngModel)]



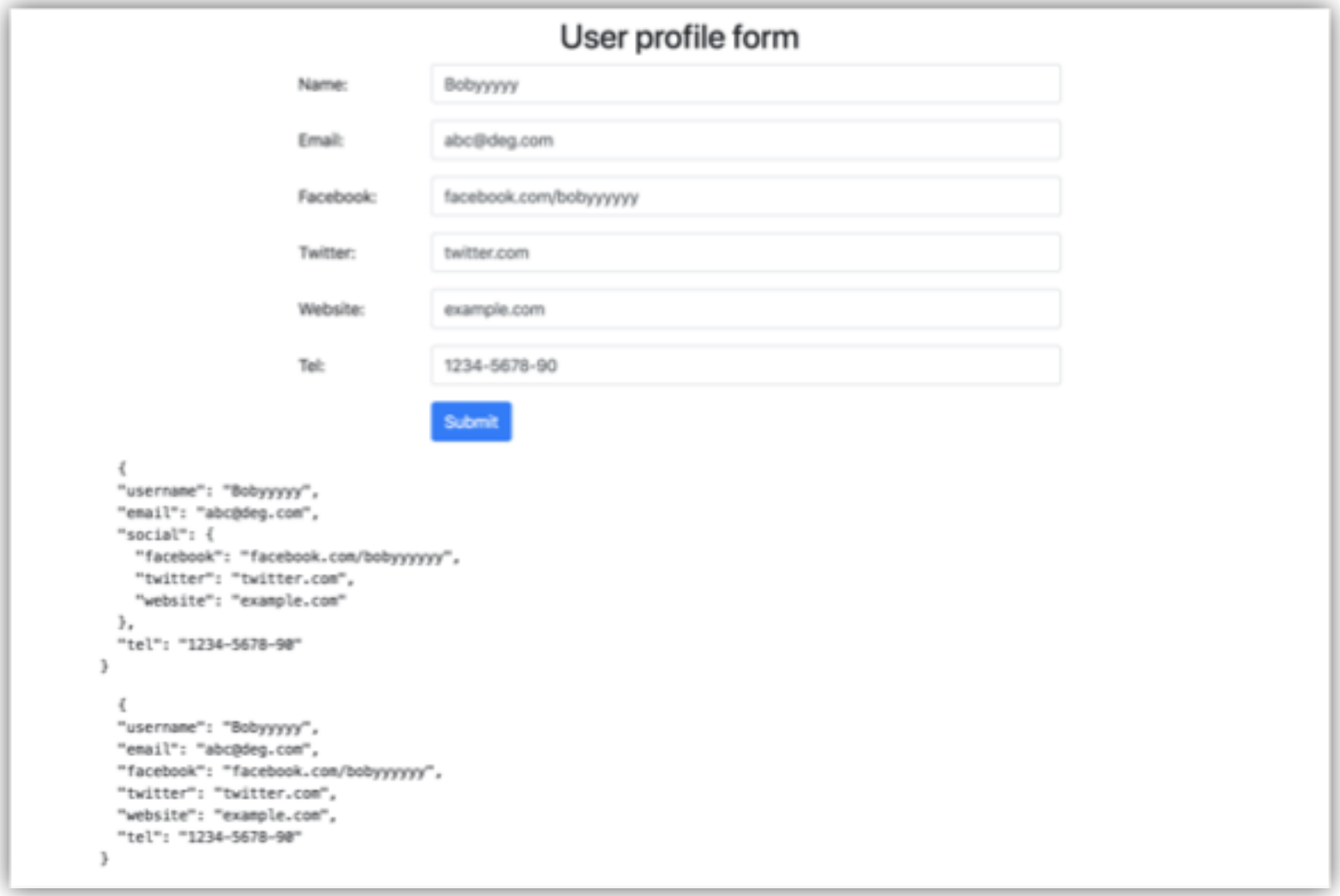
ngModelGroup directive

- Cho phép gom nhóm các control thành một nhóm – object chứa object khác.

ngModelGroup directive

```
<fieldset ngModelGroup="social">
  <input [(ngModel)]="profile.facebook"
    name="facebook">
  <input [(ngModel)]="profile.twitter"
    name="twitter">
  <input [(ngModel)]="profile.website"
    name="website">
</fieldset>
```

ngModelGroup directive



(submit) vs (ngSubmit)

- Giả sử method lắng nghe sự kiện submit form phát sinh một exception, liệu rằng form của chúng ta còn hoạt động?

```
onSubmit(profileForm) {
  // unhandle exception
  throw new Error('Something went wrong');
}
```

(submit) vs (ngSubmit)

- Lúc này form sẽ submit như một form **thông thường.**
- ngSubmit sẽ ngăn chặn việc form submit kể cả có sinh ra exception.
- ngSubmit thường được khuyên dùng thay cho submit thông thường.
- Chúng ta chỉ cần thay (submit) bằng (ngSubmit).

Template-driven validation

```
<input [(ngModel)]="profile.username"
  name="username" #username="ngModel"
  required
  minlength="6"
  class="form-control" type="text">
```

Template variable để sử dụng form control

HTML5 validation attribute, Angular sẽ tạo các directive tương ứng để thực hiện validate dữ liệu

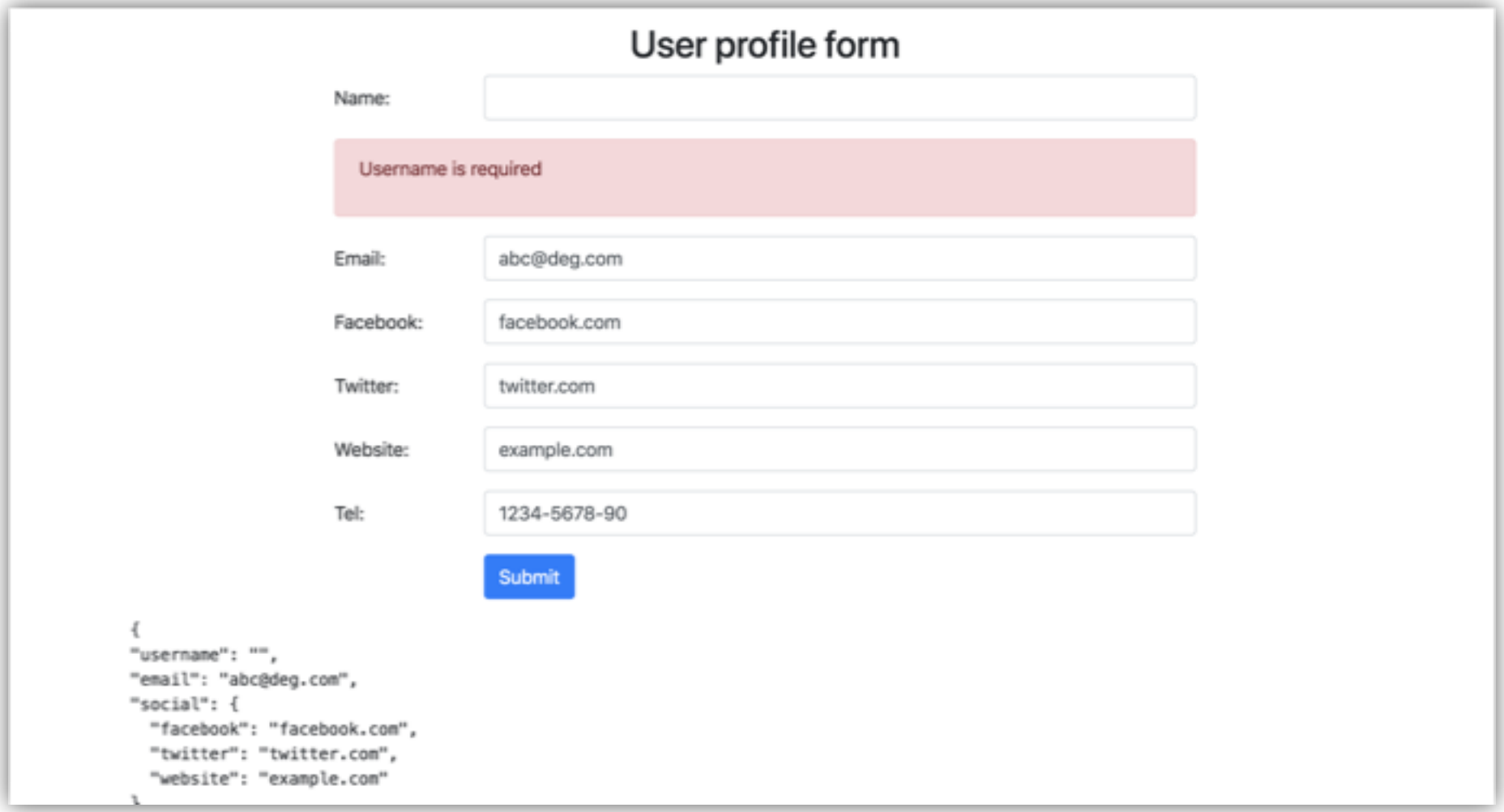
Template-driven validation

```
<div *ngIf="!username.valid && username.touched"
  class="alert alert-danger" role="alert">
  <p *ngIf="username?.errors?.required">
    Username is required
  </p>
  <p *ngIf="username?.errors?.minlength">
    The input must be of minimum length 6 characters
  </p>
</div>
```

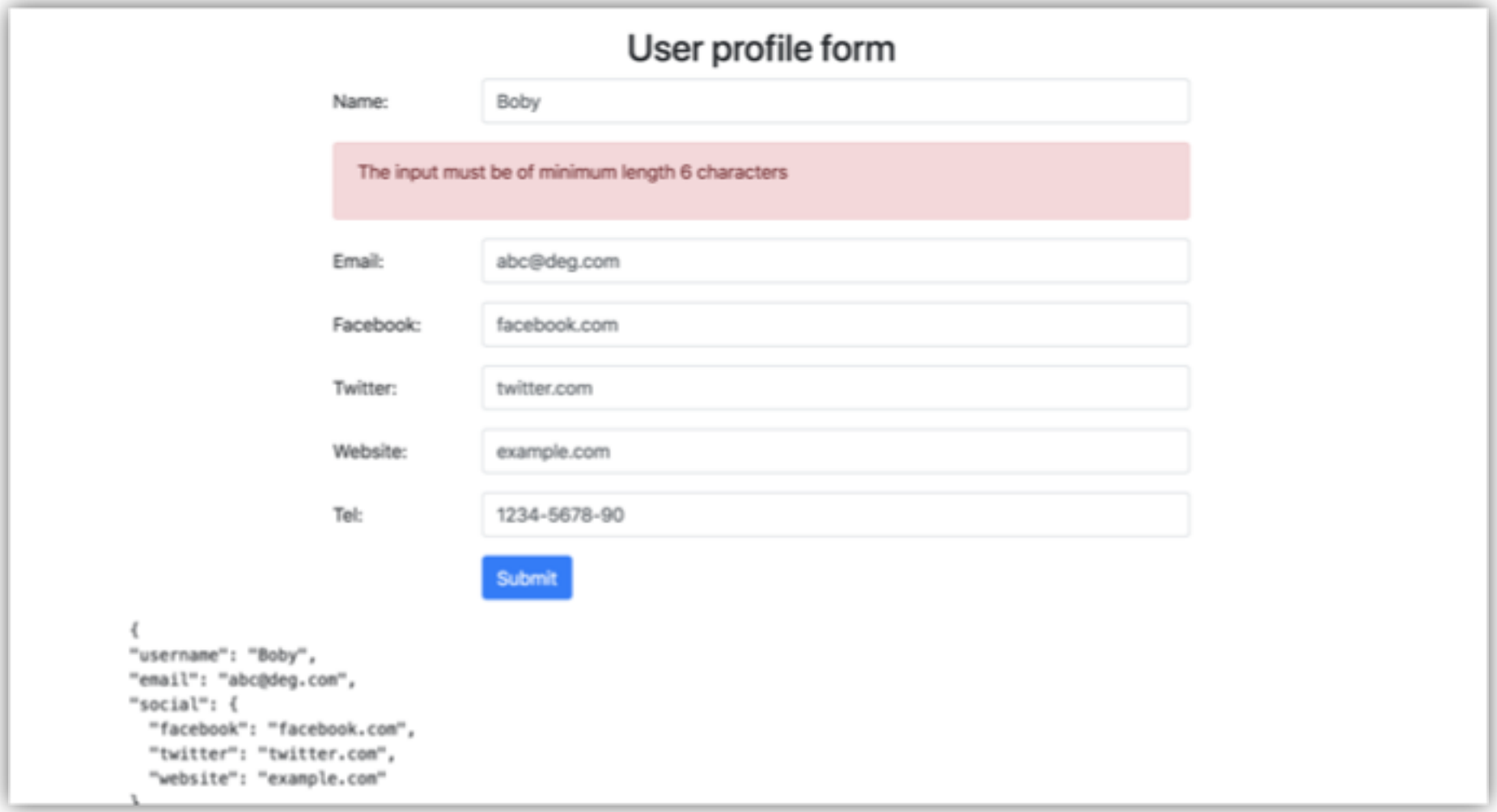
Sử dụng template variable đã tạo trước đó để kiểm tra các thông tin, nhằm hiển thị trực quan cho người dùng biết

Sử dụng safe navigation operator để tránh lỗi khi truy cập thuộc tính của null hoặc undefined

Template-driven validation



Template-driven validation



Validation state

- touched: true nếu người dùng đã focus vào input rồi thoát focus.
- untouched: true nếu người dùng chưa đụng chạm gì hoặc lần đầu tiên focus và chưa bị mất focus (ngược lại với touched)
- dirty: true nếu người dùng đã tương tác với control – nhập một ký tự vào input text chẳng hạn.
- pristine: true nếu người dùng chưa tương tác gì với control, mặc dù có thể đã touched, nhưng chưa sửa đổi gì.

Reactive Form

Reactive 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, ngForm, etc, 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.

Reactive Form

- Trong Reactive forms, chúng ta tạo toàn bộ form control tree ở trong Component (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.

Reactive Form

- Trong Template-driven forms, chúng ta ủy thác việc tạo form control cho directives, để tránh bị lỗi **changed after checked**, directives cần một cycle nữa để build toàn bộ form control tree. Vậy nên bạn cần đợi một tick nữa để có thể truy cập vào các phẩn tử của form. Chính điều này khiến việc test template-driven form trở nên phức tạp hơn. (Thường xuyên phải sử dụng **safe navigation operator**).

Reactive Form

- Trong Angular, developer thường được khuyến cáo sử dụng Reactive Form thay cho Template-driven Form để tận dụng tối đa các tính năng nâng cao của Reactive Form (lắng nghe các event khi data, state thay đổi chẳng hạn).

Import thư viện

```
import { ReactiveFormsModule } from '@angular/forms';


@NgModule({
  imports: [
    // ...
    ReactiveFormsModule
  ],
  // ...
})
export class AppModule { }
```

Khởi tạo form

```
class RegisterComponent implements OnInit {
  registerForm: FormGroup;

  ngOnInit() {
    this.registerForm = new FormGroup({
      email: new FormControl(''),
      password: new FormControl(''),
      confirmPassword: new FormControl('')
    });
  }
}
```

Nested Form

```
this.registerForm = new FormGroup({
   email: new FormControl(''),
   pwGroup: new FormGroup({
      password: new FormControl(''),
      confirmPassword: new FormControl('')
   })
});
```

Nested Form – Binding control

```
<fieldset  formGroupName="pwGroup">
   <input  formControlName="password">
</fieldset>
```

Sử dụng formGroupName để binding tương ứng với group mà chúng ta có ở trong form

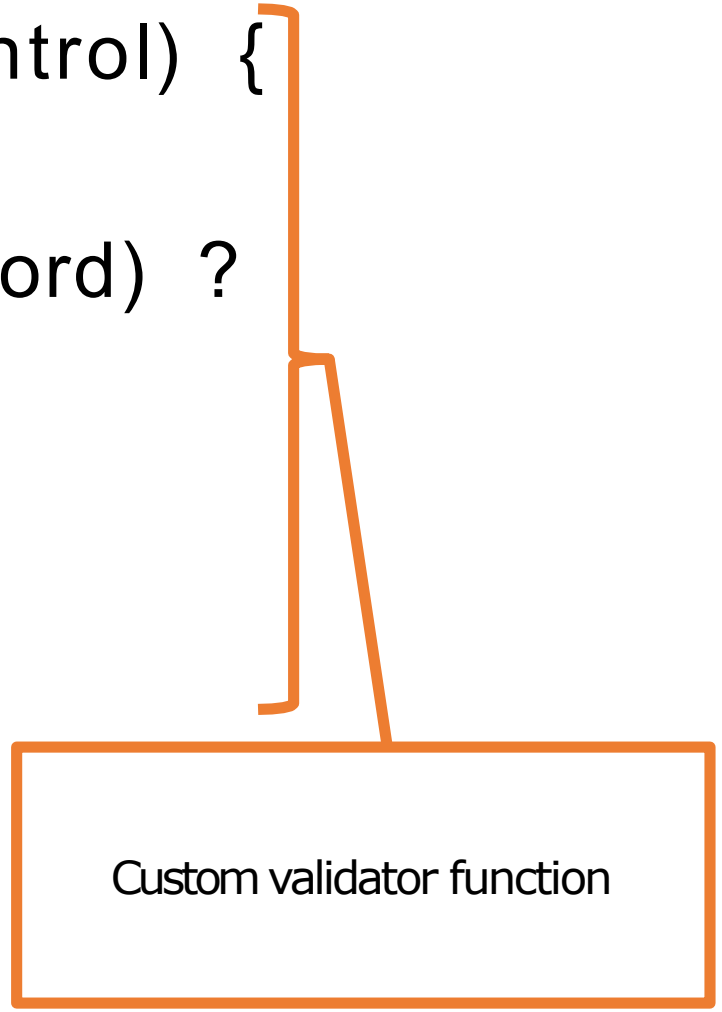
password lúc này là con của group pwGroup, không phải registerForm nữa

Reactive Form validation

- Có thể sử dụng Validator function hoặc directive.
- Rất dễ dàng để tạo custom validator function.

Reactive Form validation

```
function comparePassword(c: AbstractControl) {
  const v = c.value;
  return (v.password === v.confirmPassword) ?
    null : {
      passwordnotmatch: true
    };
}
```



Reactive Form validation

```
this.registerForm = new FormGroup({
  email: new FormControl('',
    [Validators.required, Validators.email]),
  pwGroup: new FormGroup({
    password: new FormControl(''),
    confirmPassword: new FormControl('')
  }, comparePassword)
});
```

Kết hợp các Validator function lại để đạt được hiệu quả mong muốn

Dễ dàng validate được nhiều Form Control có quan hệ với nhau

Reactive Form validation

```
<div class="alert alert-danger" role="alert"
  *ngIf="registerForm.get('email').invalid &&
    registerForm.get('email').touched">
  Invalid email!
</div>
<div class="alert alert-danger" role="alert"
  *ngIf="registerForm.hasError('passwordnotmatch',
    ['pwGroup']) && registerForm.get('pwGroup').touched">
  Password does not match!
</div>
```

Không cần sử dụng safe navigation operator ở template

Reactive Form – Form Builder

- Form Builder giúp dễ dàng tạo form.
- Sử dụng cú pháp đơn giản hơn.

Reactive Form – Form Builder

```
constructor(private fb: FormBuilder) { }
this.registerForm = this.fb.group({
  email: ['',
    [Validators.required, Validators.email]],
  pwGroup: this.fb.group({
    password: '',
    confirmPassword: ''
  }, {validator: comparePassword})
});
```

Inject service, class này cung cấp các builder method để dễ dàng tạo control

Cập nhật state cho form hoặc control

- Có 2 phương thức để cập nhật giá trị cho form control được mô tả bởi class AbstractControl là **setValue** và **patchValue**.
- Chúng là các abstract method, vậy nên các class dẫn xuất sẽ phải implement riêng cho chúng.
- Class FormControl, không có gì khác biệt giữa 2 phương thức – thực chất patchValue gọi lại setValue

Cập nhật state cho form hoặc control

- Đối với các class FormGroup và FormArray, **patchValue** sẽ cập nhật các giá trị được khai báo tương ứng trong object value truyền vào.
- Nhưng **setValue** sẽ báo lỗi nếu một control nào bị thiếu hoặc thừa, tức là bạn phải truyền chính xác object có cấu trúc giống như cấu trúc của form hay nói cách khác là không chấp nhận subset hoặc superset của cấu trúc form hiện tại.

Cập nhật state cho form hoặc control

- Nếu bạn muốn cập nhật một phần của form thì hãy dùng **patchValue**, nếu bạn muốn set lại tất cả và đảm bảo không cái nào bị thiếu thì dùng **setValue** để tận dụng việc báo lỗi của nó.
- Phương thức **reset** để bạn có thể reset lại trạng thái lúc khởi tạo của form hoặc control.

Cập nhật state cho form hoặc control

```
//  update form state
this.registerForm.patchValue({
   email: 'info@example.com'
});
```

Raising the bar