
# Forms

# Agenda

- Template-driven form
- Реактивні форми

# Template-driven

- використовуються для додавання простих форми до програми, наприклад, форми реєстрації, логіну і тд. Їх легко додати до програми, але вони не масштабуються

```
@Component({
  selector: 'app-form',
  template: `
    Favorite Color: <input type="text" [(ngModel)]="favoriteColor">
  `
})
export class FavoriteColorComponent {
  favoriteColor = '';
}
```

```
import { FormsModules } from “@angular/forms”;
```

Demo

https://angular.io/guide/forms

# Створити шаблон форми

```
<form>
  <div class="form-group">
    <input type="email" class="form-control" placeholder="Enter email">
  </div>
  <div class="form-group">
    <input type="password" class="form-control" placeholder="Password">
  </div>
  <div class="form-group">
    <select id="select" class="form-control">
      <option value="">Male</option>
      <option value="">Female</option>
    </select>
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
 <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

# Зв’язування даних

1. Створити об’єкт в компоненті

```
gender: string[] = ['Male', 'Female'];
user = {
   email: '',
   password: '',
   gender: ''
 }
```

1. Прив'язка властивостей даних до кожного елемента форми з використанням ngModel - двостороннього синтаксису прив'язки даних, дати name до кожного поля

```
<input type="email" class="form-control" placeholder="Enter email"
[(ngModel)]="user.email" name="email">
```

# Стан форми

1. Метод відправки даних

   ```
   <form (ngSubmit)="onSubmit()">
   ```

2. Зміна стану форми

   ```
   <form #regForm="ngForm" (ngSubmit)="onSubmit()">
   ```

3. Отримання стану форми

   ```
   regForm.form.valid
   ```

# Валідація

1. Додаєм атрибут required для об’язкових полів

2. Створюєм змінну шаблону і прив’язуєм до неї дані з форми
   `#email="ngModel"`
3. До поля додаються спеціальні css класи, що вказують на стан його

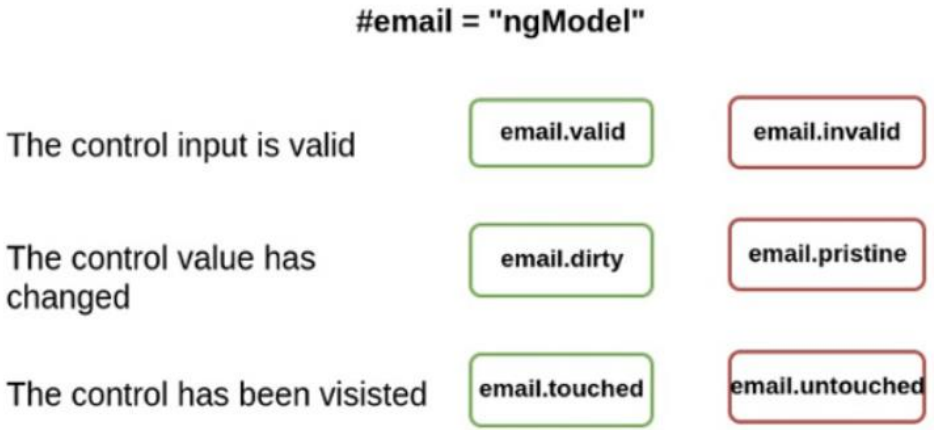
# Класи валідації

| State | Class if true | Class if false |
| --- | --- | --- |
| The control has been visited. | `ng-touched` | `ng-untouched` |
| The control's value has changed. | `ng-dirty` | `ng-pristine` |
| The control's value is valid. | `ng-valid` | `ng-invalid` |

# Повідомлення про помилки

1. Аналогічно як із класами можна отримати стан форми, але тепер у шаблонну зміну
2. Показуємо повідомлення якщо поле не валідне

```
<div [hidden]="email.valid" class="alert alert-danger">
  Name is required
</div>
```



# Приклад

[(ngModel)]="user.email" - двонаправленне звязування даних

name="email"

#regForm="ngForm" - шаблонна змінна, яка містить дані про стан форми

{{ user | json}} - актуальні значення полів

{{ regForm.form.valid | json}} - стан форми

(ngSubmit)="onSubmit()" - виклик методу при відправці форми

[disabled]="!regForm.form.valid" - коли форма буде не валідна то кнопка буде не активна

{{email.className}} - класи станів форми

#email="ngModel" - щоб можна було зробити валідацію кожного поля, потрібно додати шаблонну змінну

Demo

# Реактивні форми

- є більш надійними: вони більш масштабовані, багаторазові. Кожна зміна стану форми повертає новий стан, який підтримує цілісність моделі між змінами.

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

name = new FormControl(''); // описуємо поле

<label>   // вставляємо його в шаблон
  Name:
  <input type="text" [formControl]="name">
</label>
```

Demo

# Реалізація форми

1. групуємо поля в FormGroup

```
userForm = new FormGroup({
   email: new FormControl(''),
   password: new FormControl(''),
   gender: new FormControl(''),
 });
```

1. Зв’язуємо FormGroup з формою і визначаємо метод

```
<form [formGroup]="userForm" (ngSubmit)="onSubmit()">
    <input type="text" formControlName="email">
    <input type="password" formControlName="password">
  </form>
```

1. Отримуємо дані з форми в компоненті

```
onSubmit() {
  console.warn(this.userForm.value);
}
```

https://angular.io/guide/reactive-forms

# Модифікування значення поля

- Звертаємося до `userForm.patchValue` та визначаємо нове значення поля

```
this.userForm.patchValue({
      email: 'test@gmail.com'
      });
```

- або при ініціалізації

```
email: new FormControl('test@gmail.com'),
```

# Валідація

1. Задання обов’язкового поля

```
email: new FormControl('', Validators.required),
```

2. Визначаємо геттери кожного поля форми в компоненті.

```
get email() {
   return this.userForm.get('email');
}

get f() { return this.userForm.controls; } або зразу для всіх полів
```

1. Показуємо повідомлення про помилку

```
<div [hidden]="email.valid || email.pristine" class="alert alert-danger">
     Email is required
</div> або доступ f.email.valid якщо гетнули зразу всі поля
```

Demo

# Form Builder

- альтернативний підхід до створення форм

https://angular.io/guide/reactive-forms#generating-form-controls-with-formbuilder

# Посилання

https://metanit.com/web/angular2/5.1.php

https://code.tutsplus.com/tutorials/introduction-to-forms-in-angular-4-template-driven-forms--cms-29766

https://angular.io/guide/forms

https://metanit.com/web/angular2/5.5.php

https://angular.io/guide/reactive-forms

https://code.tutsplus.com/tutorials/introduction-to-forms-in-angular-4-reactive-forms--cms-29787

https://malcoded.com/posts/angular-reactive-form-validation/

https://jasonwatmore.com/post/2018/11/07/angular-7-reactive-forms-validation-example

# Відео

https://www.youtube.com/watch?v=JeeUY6WaXiA

https://www.youtube.com/watch?v=nGr3C3wbh9c&list=PLC3y8-rFHvwhwL-XH04cHOpJnkgRKykFi