Лекция 25. Angular Forms

[**Модуль FormsModule и директива NgModel**](#_7wfvp0rphqi) **3**

[Работа с атрибутами объекта в формах](#_11tv7dbajjah) 4

[Доступ к полям формы](#_uiz6g030xrr9) 4

[**Состояние модели и валидация**](#_egtxte3kt6bx) **4**

[Валидация элемента формы](#_ialftotk78yp) 5

[Директива NgForm](#_gnxt8e1kadkl) 5

[**Reactive Forms**](#_ufof7y6lrjxq) **7**

[Подключение к проекту](#_mhs190qmqjrg) 7

[Создание формы](#_3fh4a0m0g3mi) 7

[Валидация формы и отображение ошибок](#_g4k1q3ms01o2) 8

[Добавление своих валидаторов](#_1v9oaise6322) 8

[**Дополнительные материалы**](#_8e3sy27f7bgf) **10**

[**Задание для закрепления материала**](#_sij3urj98prj) **11**

# Модуль FormsModule и директива NgModel

Для взаимодействия с пользователем в веб-приложениях, как правило применяются формы. В Angular прежде чем использовать формы в компонентах, нам надо импортировать в главном модуле AppModule модуль FormsModule, который позволяет работать с формами:

**import { FormsModule } from '@angular/forms';**

@NgModule({

imports: [ BrowserModule, **FormsModule**],

declarations: [ AppComponent],

bootstrap: [ AppComponent ]

})

Кроме того, в файле конфигурации приложения package.json среди списка используемых зависимостей должен быть указан пакет "angular/forms":

"dependencies": {

"@angular/forms": "~6.0.0",

},

При работе с формами ключевым моментом является использование директивы NgModel. Эта директива с помощью переданной модели создает объект FormControl и привязывает эту модель к созданному элементу формы. Объект FormControl отслеживает значение модели, а также отвечает за валидацию этого значения и взаимодействие с пользователем.

Данная директива принимает переданную ей модель в качестве входного свойства. Причем мы можем использовать как однонаправленную, так и двунаправленную привязку.

Если нам надо просто вывести значение модели в поле ввода, то можно ограничиться и однонаправленной привязкой:

**<input name="title" [ngModel]="title" />**

Это обычная привязка свойства, где в качестве модели используется некоторое свойство title, определенное в классе компонента.

Если нам надо отслеживать изменение введенных данных, то мы можем использовать двунаправленную привязку:

**<input name="title" [(ngModel)]="title" />**

## Работа с атрибутами объекта в формах

Для упрощения работы с объектами в моделях мы можем применять привязку не только к переменной в компоненте, но и к атрибуту объекта:

**<label>**

**Марка:**

**<input type=”text” [(ngModel)]=”car.brand”>**

**</label>**

**<label>**

**Модель:**

**<input type=”text” [(ngModel)]=”car.model”>**

**</label>**

**<label>**

**Цена:**

**<input type=”text” [(ngModel)]=”car.price”>**

**</label>**

## Доступ к полям формы

Помимо доступа к значению переменной через ngModel имеется возможность получить доступ к полям через атрибут id:

<input type=”text” name=”price” [(ngModel)]=”car.price” #carPrice />

{{carPrice.name}} : {{carPrice.model}}

# Состояние модели и валидация

Помимо связи с переменной компонента, ngModel предоставляет информацию о состоянии инпута с помощью классов:  
  
**ng-untouched -** устанавливается, если элемент формы не получал фокуса.

**ng-touched** - устанавливается, если элемент формы получал фокус.

**ng-dirty -** устанавливается, если значение элемента формы было изменено вручную.

**ng-pristine -** устанавливается, если значение элемента не изменялось с момента загрузки страницы.

**<input class="form-control ng-untouched ng-pristine ng-valid" name="title" ng-reflect-name="title" />**

## Валидация элемента формы

Angular предоставляет расширенные возможности для валидации формы и вывода сообщений об ошибках:

<div class="form-group">

<label>Имя</label>

<input class="form-control" name="name" [(ngModel)]="user.name" #name="ngModel" required />

<div [hidden]="name.valid || name.untouched" class="alert alert-danger">

Не указано имя

</div>

</div>

<div class="form-group">

<label>Email</label>

<input class="form-control" name="email" [(ngModel)]="user.email" #email="ngModel" required pattern="[a-zA-Z\_]+@[a-zA-Z\_]+?\.[a-zA-Z]{2,3}" />

<div [hidden]="email.valid || email.untouched" class="alert alert-danger">

Некорректный email

</div>

</div>

Для того, чтобы валидация была полной, необходимо отключить возможность отправки формы:  
  
<button [disabled]="name.invalid || email.invalid || phone.invalid"

class="btn btn-default" (click)="addUser()">Добавить</button>

## Директива NgForm

Как правило, элементы формы не используются без формы как таковой. В Angular предусмотрена директива NgForm. Форма обладает такими же состояниями как и элементы внутри нее:  
- ng-pristine

- ng-untouched

- ng-valid

Определяется форма следующим образом:

<form #formName="ngForm" novalidate>

с того момента, как для формы был добавлен атрибут id, она становится доступной внутри компонента:  
  
submit(){

console.log(formName);

}

также она становится доступной как элемент валидации:

<input type=”submit” [disabled]=”formName.invalid || ….” />

# Reactive Forms

Подход работы с формой, рассмотренный в предыдущих разделах, основанный на работе с шаблонами называется Template Driven. Существует второй подход генерации формы, называемый Reactive Forms, он основан на генерации формы из Компонента (кода).

## Подключение к проекту

Для использования Reactive Forms необходимо добавить модуль к описанию проекта:

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

**import { FormsModule } from '@angular/forms';**

import { AppComponent } from './app.component';

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

@NgModule({

imports: [ BrowserModule, **FormsModule, ReactiveFormsModule**],

declarations: [ AppComponent],

bootstrap: [ AppComponent ]

})

export class AppModule { }

## Создание формы

Для использования Reactive Forms в компоненте необходимо указать в блоке импорта следующее:

import { FormGroup, FormControl, Validators} from '@angular/forms';

Само создание форму в компоненте выглядит следующим образом:

myForm : FormGroup = new FormGroup({

"brand": new FormControl(),

"model": new FormControl(),

"price": new FormControl(),

});

после создания кода формы в компоненте, мы можем связать шаблон с моделью:

<form [formGroup]="myForm" >

<input name="brandName" formControlName="brand">

<input name="modelName" formControlName="model">

<input name="price" formControlName="price">

</form>

При создании элементов формы можно задать значение по умолчанию и набор валидаторов:

myForm : FormGroup = new FormGroup({

"brand": new FormControl(“BMW”, [

Validators.required,

Validators.pattern(“(a-zA-Z)+”)

]),

"model": new FormControl(),

"price": new FormControl(),

});

В случае с элементом Brand, мы указываем значение по умолчанию и

валидатора: required и pattern.

## Валидация формы и отображение ошибок

Вывод ошибок валидации может быть выполнен в форме:

<div [hidden]=”!**myForm.controls[‘brandName’].valid**”>

<div \*ngIf=”**myForm.controls[‘brandName’].required**”>

Поле “Марка машины” должно быть заполнено

</div>

<div \*ngIf=”**myForm.controls[‘brandName’].pattern**”>

Поле “Марка машины” заполнено не по формату

</div>

</div>

## Добавление своих валидаторов

В случае, если нам необходимы дополнительные проверки, мы можем добавить свои валидаторы:

myForm : FormGroup = new FormGroup({

"brand": new FormControl(“BMW”, [

Validators.required,

Validators.pattern(“(a-zA-Z)+”),

**this.existingCarBrand**

]),

"model": new FormControl(),

"price": new FormControl(),

});

**existingCarBrand**(control: FormControl): {[s:string]:boolean}{

let existingBrands = [‘BMW’, ‘Mersedes’, ‘Lada’, ‘Toyota’];

if(!existingBrands.indexOf( control.value ) === -1){

return {"brand": true};

}

return null;

}

# Дополнительные материалы

<https://metanit.com/web/angular2/>

# Задание для закрепления материала

Разработать механизм переключения сложности игры:

1. Простой уровень. Компьютерный противник стреляет наугад, вне зависимости от того, был ли корабль игрока ранен.
2. Средний: Компьютер пытается добить корабль игрока и не стреляет в те координаты, где корабля игрока быть не может (например, впритык к уничтоженному кораблю)
3. Сложный: Компьютер пытается вычислить наиболее крупные корабли игрока с помощью стрельбы крест-накрест (a1 - к10, к10-а1). В случае полного прохождения диагоналей, компьютер простреливает поле в шахматном порядке. Все замечания из предыдущего пункта (средний) должны быть применены и к сложному варианту