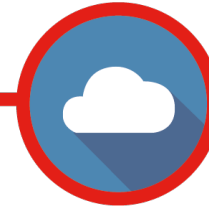


Forms



Forms

- Angular's form modules
- Validation

Please Enter Your Phone Number:

216 ▼ 410 ▼ 0000 ▼

Next

0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019 ▼

Elements Console Sources Network Timeline Profile

```
<option value="0481">0481</option>
<option value="0482">0482</option>
<option value="0483">0483</option>
<option value="0484">0484</option>
<option value="0485">0485</option>
<option value="0486">0486</option>
<option value="0487">0487</option>
<option value="0488">0488</option>
<option value="0489">0489</option>
<option value="0490">0490</option>
<option value="0491">0491</option>
<option value="0492">0492</option>
<option value="0493">0493</option>
<option value="0494">0494</option>
<option value="0495">0495</option>
<option value="0496">0496</option>
<option value="0497">0497</option>
<option value="0498">0498</option>
<option value="0499">0499</option>
</select>
<br>
<br>
<button id="next" class="btn btn-
primary">Next</button>
::after
</div>
</body>
<div id="coFrameDiv" style="height:0px;display:none;"
- </div>
</html>
```

html body div.container select

Forms - Two styles

- Template Driven
 - `FormsModule`
- Model Driven
 - `ReactiveFormsModule`

Reactive Forms

- Observable streams
- Angular style
- Angular classes



Reactive Forms

```
@NgModule({  
  imports: [ ReactiveFormsModule ]  
})
```

Reactive Forms - classes

- FormControl
- FormGroup
- FormArray
- FormBuilder

Reactive Forms

FormGroup & FormControl

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl()
  });

}
```

```
<form [formGroup]="contactForm">

  <input
    formControlName="name" />

</form>
```

Reactive Forms

FormGroup & FormControl

```
@Component({ selector: 'contacts' })  
export class ContactsComponent {  
  
  contactForm = new FormGroup({  
    name: new FormControl()  
  });  
  
}
```

```
<form [formGroup]="contactForm">  
  
  <input  
    formControlName="name" />  
  
</form>
```


Reactive Forms

FormGroup & FormControl

```
@Component({ selector: 'contacts' })  
export class ContactsComponent {  
  
  contactForm = new FormGroup({  
    name: new FormControl()  
  });  
  
}
```

```
<form [formGroup]="contactForm">  
  
  <input  
    formControlName="name" />  
  
</form>
```

Reactive Forms - events

- (ngSubmit)
- .valueChanges
- .statusChanges
- updateOn

Reactive Forms

Submit

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl()
  });

  submit() {
    console.log(
      this.contactForm.value
    );
  }
}
```

```
<form [formGroup]="contactForm"
      (ngSubmit)="submit()">

  <input
    formControlName="name" />

  <button type="submit">
    Send
  </button>

</form>
```

Reactive Forms

Submit

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl()
  });

  submit() {
    console.log(
      this.contactForm.value
    );
  }
}
```

```
<form [formGroup]="contactForm"
      (ngSubmit)="submit()">

  <input
    formControlName="name" />

  <button type="submit">
    Send
  </button>

</form>
```

Reactive Forms - validators

- `Validators.required`
- `Validators.minLength(i)`
- `Validators.pattern(/ /)`
- `(c: FormControl): ValidationErrors |
null => { }`

Reactive Forms

This is what we had...

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl()
  });

  submit() {
    console.log(
      this.contactForm.value
    );
  }
}
```

```
<form [formGroup]="contactForm"
      (ngSubmit)="submit()">

  <input
    formControlName="name" />

  <button type="submit">
    Send
  </button>

</form>
```

Reactive Forms

... now with validators

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl('',
      [ Validators.required,
        Validators.minLength(3) ]
    )
  });

  get name() { return
    this.contactForm.get('name')
  }

}
```

```
<form [formGroup]="contactForm"
      (ngSubmit)="submit()">

  <input
    FormControlName="name" />

  <span *ngIf="name.invalid">
    Name is required.
  </span>

</form>
```

Reactive Forms

Validators

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl('',
      [ Validators.required,
        Validators.minLength(3) ]
    )
  });

  get name() { return
    this.contactForm.get('name')
  }

}
```

```
<form [formGroup]="contactForm"
      (ngSubmit)="submit()">

  <input
    formControlName="name" />

  <span *ngIf="name.invalid">
    Name is required.
  </span>

</form>
```


Reactive Forms

getters as shortcut for readability

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl('',
      [ Validators.required,
        Validators.minLength(3)]
    )
  });

  get name() { return
    this.contactForm.get('name')
  }

}
```

```
<form [formGroup]="contactForm"
      (ngSubmit)="submit()">

  <input
    formControlName="name" />

  <span *ngIf="name.invalid">
    Name is required.
  </span>

</form>
```

Reactive Forms - form builder

More extensive forms

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm = new FormGroup({
    name: new FormControl('',
      [ Validators.required, Validators.minLength(3) ]
    ),
    surname: new FormControl('',
      [ Validators.required, Validators.minLength(1) ]
    ),
    surnameprefix: new FormControl(''),
    address: new FormGroup({
      zipcode: new FormControl('',
        [ Validators.required, Validators.pattern(/.*/)],
        [ validZipCodeBackendValidator ]
      ),
      street: new FormControl('',
        [ Validators.required, Validators.pattern(/.*/)]
      )
    }),
    birthdate: new FormControl('',
      [ Validators.required, Validators.minLength(1) ]
    ),
  });
}
```

Reactive Forms - form builder

Shorter notation

```
@Component({ selector: 'contacts' })
export class ContactsComponent {

  contactForm: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.contactForm = this.fb.group({
      name: ['', [ Validators.required, Validators.minLength(3) ] ],
      surname: ['', [ Validators.required, Validators.minLength(1) ] ],
      surnameprefix: '',
      address: this.fb.group({
        zipcode: ['', [ Validators.required, Validators.pattern(/.*/) ] ],
        street: ['', [ Validators.required, Validators.pattern(/.*/) ] ],
      }),
      birthdate: ['', [ Validators.required, Validators.minLength(1) ] ]
    });
  }
};
```

Recap: Reactive Forms

- FormGroup, FormControl
- Events
- Validation
- FormBuilder

Demo

Please Enter Your Phone Number:

216 ▼

410 ▼

0000 ▼
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019 ▼

Next



```
<option value="0481">0481</option>
<option value="0482">0482</option>
<option value="0483">0483</option>
<option value="0484">0484</option>
<option value="0485">0485</option>
<option value="0486">0486</option>
<option value="0487">0487</option>
<option value="0488">0488</option>
<option value="0489">0489</option>
<option value="0490">0490</option>
<option value="0491">0491</option>
<option value="0492">0492</option>
<option value="0493">0493</option>
<option value="0494">0494</option>
<option value="0495">0495</option>
<option value="0496">0496</option>
<option value="0497">0497</option>
<option value="0498">0498</option>
<option value="0499">0499</option>
Show All Nodes (9499 More)
</select>
<br>
<br>
<button id="next" class="btn btn-
primary">Next</button>
::after
</div>
</body>
</html>
```

html body div.container select

FormsModule

- Mostly from the template
- Targeted at AngularJS developers
- Less "DSL"

FormsModule

```
@NgModule({  
  imports: [ FormsModule ]  
})
```

NgForm

NgForm is automatically bound to <form>

```
<form>  
  <input ...>  
</form>
```


Template reference variable

#var makes a variable of any element

```
<form #newContactForm>  
  <input ...>  
  {{ newContactForm.action }}  
</form>
```

Template reference variable

#var="ngForm" provides a variable with an instance of NgForm

```
<form #newContactForm="ngForm">  
  <input ...>  
  
  <button type="submit"  
    [disabled]="newContactForm.invalid">  
  </button>  
  
</form>
```

NgForm properties

- valid | invalid
- hasError('required', 'fieldname')
- touched | untouched
- dirty | pristine
- submitted
- resetForm()

Connecting fields

ngModel connects the field to NgForm

```
<form #newContactForm="ngForm">  
  <input ngModel name="name">  
  
  <button type="submit"  
    [disabled]="newContactForm.invalid">  
  </button>  
  
</form>
```

Template reference variable

works on almost any directive

```
<form #newContactForm="ngForm">  
  <input ngModel name="name" #name="ngModel">  
  <button type="submit"  
    [disabled]="newContactForm.invalid">  
  </button>  
</form>
```

Adding validation

Angular supports HTML5 validators

```
<form #newContactForm="ngForm">

  <input ngModel name="name" #name="ngModel" required>

  <button type="submit"
    [disabled]="newContactForm.invalid">
  </button>

</form>
```

Showing field status

Get the state of the field from NgModel

```
<form #newContactForm="ngForm">  
  <input ngModel name="name" #name="ngModel" required>  
  <div *ngIf="name.invalid"> Name is required </div>  
  <button ...> </button>  
</form>
```

ngModel properties

- valid, invalid
- errors, hasError('required')
- touched | untouched
- dirty | pristine

HTML5 validators

- required
- pattern
- minlength
- (maxlength)

Send the form

(ngSubmit) also sets "submitted" flag on NgForm

```
<form #newContactForm="ngForm"  
  (ngSubmit)="submit(newContactForm)">  
  <input ngModel name="name">  
  
  <button type="submit"  
    [disabled]="newContactForm.invalid">  
  </button>  
  
</form>
```

ngModel CSS

- ng-valid | ng-invalid
- ng-touched | ng-untouched
- ng-dirty | ng-pristine

FormsModule - pro's

- FormsModule
- Comparable to AngularJS
- ngModel for data binding
- Validation mostly in HTML

FormsModule - cons

- Unit tests are "harder"
- Validation logic in templates
- Works best for about ~5 fields
- Less control

Reactive Forms - pro's

- Validation logic in class
- Easier to unit test
- More control
- Observable streams
- >5 fields

Reactive Forms - cons

- More Angular specific
- FormBuilder, FormGroup, FormArray, FormControl
- Observable streams