

Angular 2

Module 7 – Forms



Peter Kassenaar –
info@kassenaar.com



Contents

- Form Fundamentals
- Template Driven Forms
- Reactive Forms (aka *Model Driven Forms*)
- Subscribing to Form events



Forms in Web Applications - Tasks

- Initialize Default Values

A screenshot of a web form element, specifically an email input field. The field is a rectangular box with a thin black border. Inside the box, the text 'peter@kassenaar.nl' is entered in a light blue font. Below the input field, there is a line of smaller, grey text that reads 'We'll never share your email with anyone else.'



Forms in Web Applications - Tasks

- Initialize Default Values
- Validate Data

Enter email

We'll never share your email with anyone else.



Forms in Web Applications - Tasks

- Initialize Default Values
- Validate Data
- Display Validation messages

Angular Connect ID

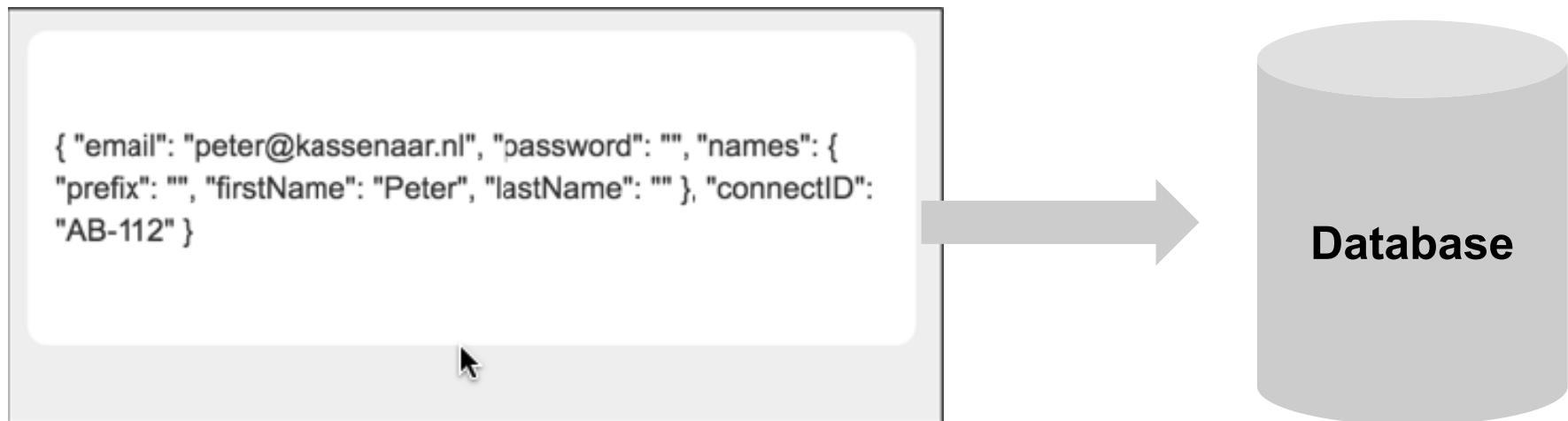
Connect ID

Please enter ID in the pattern AA-XX where A is letter and X is a number



Forms in Web Applications - Tasks

- Initialize Default Values
- Validate Data
- Display Validation messages
- **Serialize User Data**





Forms in Web Applications - Tasks

- Initialize Default Values
- Validate Data
- Display Validation messages
- Serialize User Data
- Dynamic Forms & Dynamic Controls

```
{
  key: 'email',
  type: 'input',
  templateOptions: {
    type: 'email',
    label: 'Email address',
    placeholder: 'Enter email'
  }
},
{
  key: 'password',
  type: 'input',
  templateOptions: {
    type: 'password',
    label: 'Password',
    placeholder: 'Password'
  }
},
}
```

A web form with a green border. It contains the following fields: 'FULL NAME' with a text input; 'EMAIL ADDRESS' with a text input containing 'Example: john@gmail.com'; 'SEX' with radio buttons for 'Male' and 'Female'; 'DATE OF BIRTH' with a date picker showing '--' and a calendar icon; and 'TIME OF ARRIVAL' with a dropdown menu showing 'Please select' and a downward arrow.



Forms in Web Applications - Tasks

- Initialize Default Values
- Validate Data
- Display Validation messages
- Serialize User Data
- Dynamic Forms & Dynamic Controls
- Custom Controls & Custom Validation

Search the table ...							
	Inv No	Date	Name	Amount	Price	Cost	Note
690	Inv No 690	7/15/2012	Name 690	444	671	297924	Note 690
691	Inv No 691	7/15/2012	Name 691	657	865	568305	Note 691
692	Inv No 692	7/15/2012	Name 692	804	92	73968	Note 692
693	Inv No 693	7/15/2012	Name 693	625	135	84375	Note 693
694	Inv No 694	7/15/2012	Name 694	906	608	550848	Note 694
695	Inv No 695	7/15/2012	Name 695	360	393	141480	Note 695
696	Inv No 696	7/15/2012	Name 696	293	600	175800	Note 696
697	Inv No 697	7/15/2012	Name 697	166	309	51294	Note 697



Angular 2 – Types of Forms

- Template Driven Forms

- Model Driven
(Reactive Forms)



Angular 2 – Types of Forms

- Template Driven Forms

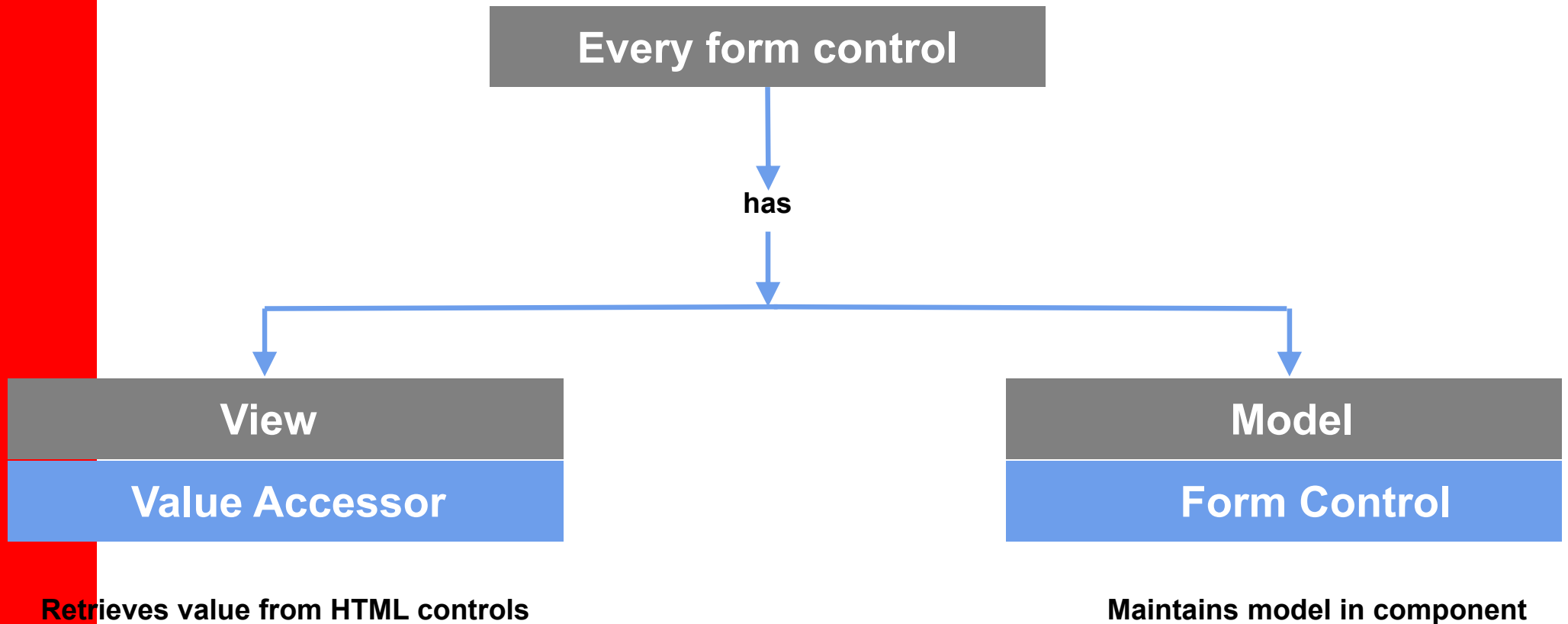
- Source of truth is the Template
- Define templates. Angular generates form model o/t fly
- Less descriptive
- Quickly Build simple forms – Less control
- Less testable

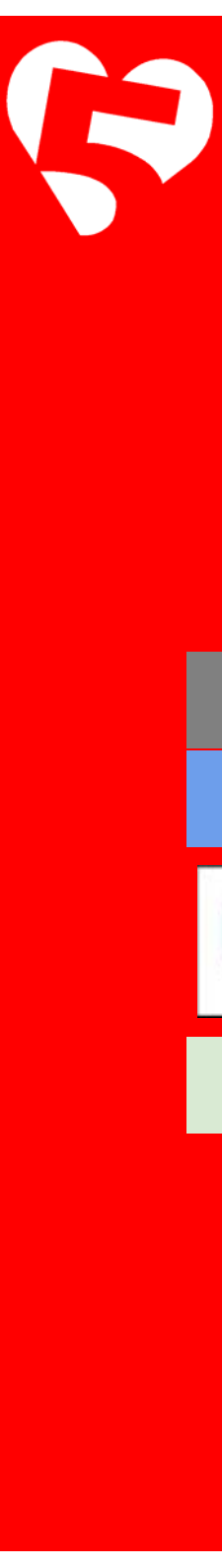
- Model Driven (Reactive Forms)

- Source of truth is the component class / directive
- Instantiate Form model and Control model yourself
- More Descriptive
- Code all the details. Takes more time, gives more control
- Very good testable



Angular 2 Forms - Fundamentals





In more detail

Every form control

View

Value Accessor

peter@kassenaar.nl

We'll never share your email with anyone else.

Default Value Accessor

Model

Form Control

M
O
D
E
L

Value

Validity

State



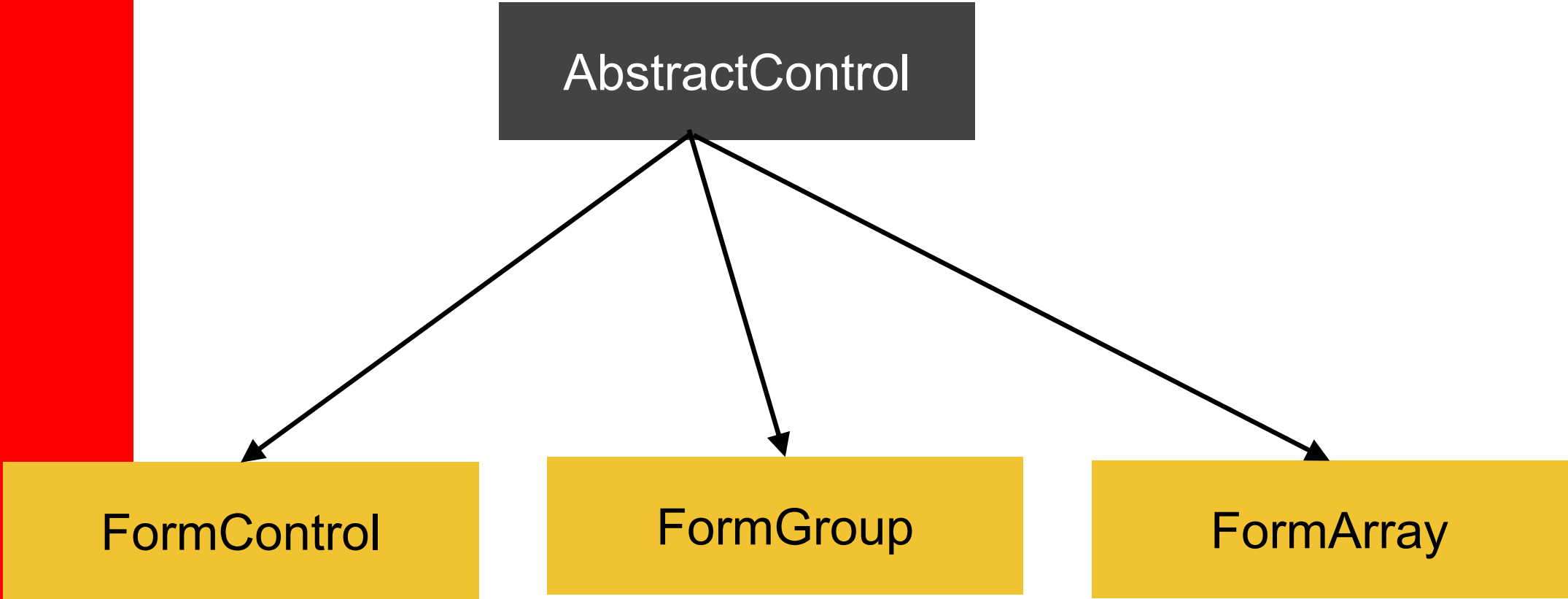
Angular 2 Forms - Base class

```
export abstract class AbstractControl {  
    _value: any;  
  
    ...  
  
    private _status: string;  
    private _errors: {[key: string]: any};  
    private _pristine: boolean = true;  
    private _touched: boolean = false;  
  
    ...  
  
    get value(): any { return this._value; }  
    get valid(): boolean { return this._status === VALID; }  
  
    ...  
  
    abstract setValue(value: any, options?: Object): void;  
  
    ...  
}
```

<https://github.com/angular/angular/blob/master/modules/%40angular/forms/src/model.ts>



Angular 2 Forms – extended classes





Control classes in code

```
653 export class FormControl extends AbstractControl {
```

```
654   /** @internal */
```

```
655   _onChange: Function[] = [];
```

```
656
```

```
657   constructor(
```

```
658     854 export class FormGroup extends AbstractControl {
```

```
659     855   constructor(
```

```
660     856     public controls: {[key: string]: AbstractControl}, validator: ValidatorFn = null,
```

```
661     857     asyncValidator: AsyncValidatorFn = null) {
```

```
662     858     super(validator, asyncValidator);
```

```
663     859     this._initObservables();
```

```
664     860     t
```

```
665     861     t
```

```
862   }
```

```
1155 export class FormArray extends AbstractControl {
```

```
1156   constructor(
```

```
1157     public controls: AbstractControl[], validator: ValidatorFn = null,
```

```
1158     asyncValidator: AsyncValidatorFn = null) {
```

```
1159     super(validator, asyncValidator);
```

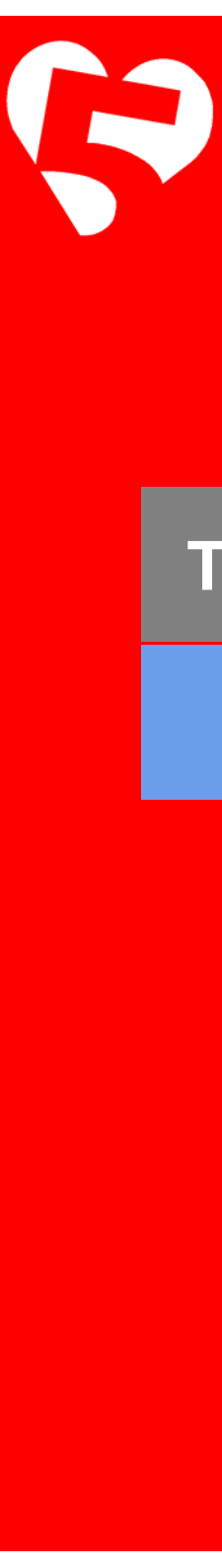
```
1160     this._initObservables();
```

```
1161     this._setUpControls();
```

```
1162     this.updateValueAndValidity({onlySelf: true, emitEvent: false});
```

```
1163   }
```

<https://github.com/angular/angular/blob/master/modules/%40angular/forms/src/model.ts>



Summary – what have we learned so far

1

2

Template Driven Forms

Less to code

Model Driven Forms

More to code

3

Model

Abstract Control,
FormControl/Group/Array



Angular 2 – Types of Forms

- Template Driven Forms

- Model Driven
(Reactive Forms)



Let's build a template driven form!

- Step 1 – Import FormsModule in app.module.ts


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






Step 2 – Add FormsModule to @ngModule

```
import {NgModule}      from '@angular/core';  
import {BrowserModule} from '@angular/platform-browser';  
import {FormsModule}   from '@angular/forms';
```



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

```
@NgModule({  
  imports      : [BrowserModule, FormsModule],  
  declarations: [AppComponent],  
  bootstrap    : [AppComponent]  
})  
  
export class AppModule {  
}
```





Step 3 – write form in HTML

```
<form novalidate>
  <div class="form-group">
    <label for="inputEmail">Email address</label>
    <input type="email" class="form-control" id="inputEmail"
      placeholder="Enter email" name="email">
    <small class="form-text text-muted">
      We'll never share your email with anyone else.
    </small>
  </div>
  <div class="form-group">
    <label for="inputPassword">Password</label>
    <input type="password" class="form-control" id="inputPassword"
      placeholder="Password" name="password">
  </div>

  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

This is just plain HTML. No Angular stuff here...



Step 4. Defining a Template Driven Form

- Add `#myForm="ngForm"` to the `<form>` tag
 - This declares a local variable with the name `#myForm` to the `<form>` element. It is of type `NgForm`
- Add `ngModel` to each and every form field
 - No value necessary

```
<form novalidate #myForm="ngForm">
  <div class="form-group">
    <input type="email" class="form-control" id="inputEmail"
      placeholder="Enter email" name="email" ngModel>
  </div>
  <div class="form-group">
    <input type="password" class="form-control" ngModel
      id="inputPassword" placeholder="Password" name="password">
  </div>
```



Just checking – Sample results pane

```
<div class="form-result">
  <h3>Validity</h3>
  <div class="validity" [ngClass]="{'invalid-form': !myForm.valid}">
    <div *ngIf="myForm.valid">Valid</div>
    <div *ngIf="!myForm.valid">Invalid</div>
  </div>
  <h3>Results</h3>
  <div class="result">
    {{ myForm.value | json }}
  </div>
</div>
```

Just to show runtime results of the Validity and Value of the form using

`myForm.valid`

`myForm.value`



Results so far

17a - Angular 2 - Template Driven Forms

Email address

We'll never share your email with anyone else.

Password

Submit

Validity

Valid

Results

```
{ "email": "", "password": "" }
```



Checkpoint

- The `#myForm` exposes the value and the validity of the form as a whole.
- `ngModel` adds the individual controls to the `#myForm`.
- You can now check it's value and state in the results pane
- Try what happens if you remove one of the `ngModel` directives!
- Check for yourself: the value of a form is a JSON-object.



Addressing individual controls



Retrieve values from individual controls

- Do the same as with the form
- Add for example #email="ngModel" to input field
- Now, the value, validity and state (i.e. its ValueAccessors!) are accessible through the local template variable

```
<label for="inputEmail">Email address</label>
```

```
<pre>value: {{ email.value }} - valid : {{ email.valid}}</pre>
```

```
<input type="email" class="form-control" id="inputEmail"  
      placeholder="Enter email" name="email" ngModel #email="ngModel">
```

```
<small class="form-text text-muted">
```

```
    We'll never share your email with anyone else.
```

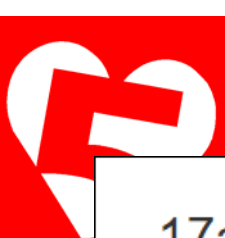
```
</small>
```



Required fields

- Add HTML5 attribute required to the input field.
- No checking on type yet!
 - It's just required.

```
<input type="email" class="form-control" id="inputEmail"  
placeholder="Enter email" name="email" ngModel #email="ngModel" required
```



17a - Template Driven Forms /app.component2.html | .ts

Email address

value: - valid : false

We'll never share your email with anyone else.

Password

Validity

Invalid

Results

```
{ "email": "", "password": "" }
```

17a - Template Driven Forms /app.component2.html | .ts

Email address

value: test - valid : true

We'll never share your email with anyone else.

Validity

Valid

Results



Angular classes and checks

- Angular adds classes to the rendered HTML to indicate state
 - ng-untouched / ng-touched,
 - ng-pristine / ng-dirty
 - ng-invalid / ng-valid

```
<h2>17a - Template Driven Forms /app.component2.html | .ts</h2>
<hr>
<form novalidate class="ng-untouched ng-pristine ng-invalid">
  <div class="form-group">
    <label for="inputEmail">Email address</label>
    <pre>value: - valid : false</pre>
    <input class="form-control ng-untouched ng-pristine" id="inputEmail" name="email" ngmodel placeholder="Enter email" type="email" ng-reflect-required="true" ng-reflect-name="email" ng-reflect-model="" />
    <small class="form-text text-muted">We'll never share your email with anyone else.</small>
  </div>
```

```
<h2>17a - Template Driven Forms /app.component2.html | .ts</h2>
<hr>
<form novalidate class="ng-dirty ng-valid ng-touched">
  <div class="form-group">
    <label for="inputEmail">Email address</label>
    <pre>value: test - valid : true</pre>
    <input class="form-control ng-dirty ng-valid ng-touched" id="inputEmail" name="email" ngmodel placeholder="Enter email" type="email" ng-reflect-required="true" ng-reflect-name="email" ng-reflect-model="test" />
    <small class="form-text text-muted">We'll never share your email with anyone else.</small>
  </div>
<div class="form-group">...
```

Write your own CSS to
define styles!



Using ngModelGroup



Adding ngModelGroup

- Combining form fields into logical groups

```
<div ngModelGroup="customer" #customer="ngModelGroup">  
  <div class="form-group">  
    ...  
  </div>  
</div>
```

Use a local template variable (i.e.

`#customer="ngModelGroup"`) only if you want to have access to the state and validity of the group as a whole.



ngModelGroup creates a nested object

17a - ngModelGroup

/app.component3.html | .ts

Email address

value: info@kassenaar.com - valid : true

info@kassenaar.com

We'll never share your email with anyone else.

Password

value: test - valid : true

....

Prefix

Mr.

First Name

Peter

Last Name

Kassenaar

```
value {  
  "namePrefix": "Mr.",  
  "firstName": "Peter",  
  "lastName": "Kassenaar"  
}  
valid: true
```

Submit

Validity

Valid

Results

```
{ "email": "info@kassenaar.com", "password": "test", "customer":  
  { "namePrefix": "Mr.", "firstName": "Peter", "lastName":  
    "Kassenaar" } }
```




Submitting forms



Define a (click) handler on the button

- Only activate the button if the form is valid
- Pass `myForm` as a parameter
- Note: no actual need for two-way databinding with `[(ngModel)]`

```
<button type="submit" class="btn btn-primary"
  (click)="onSubmit(myForm)"
  [disabled]="!myForm.valid">
  Submit
</button>
```

```
onSubmit(form){
  console.log('Form submitted: ', form.value);
  alert('Form submitted!' + JSON.stringify(form.value))
}
```



More on Template Driven Forms

The screenshot shows a dark-themed header for a user profile. On the left is a circular profile picture of a man. To the right of the picture is the text: "I'm Todd, a Developer Advocate @Telerik. Co-Founder of @UltimateAngular Creator of the Angular 2 migration guide. JavaScript, Angular, React, conference speaker. Developer Expert at Google." Below this text are two buttons: a blue "Follow @toddmotto" button and a white "29.5K followers" button. Below the header is a navigation bar with links: "Posts", "About", "Speaking", "Styleguide", "ngMigrate", and "Online training" (which has a red icon). The main content area has a light blue background. It features the title "Angular 2 form fundamentals: template-driven forms" in a large, elegant serif font. Below the title is a horizontal line, followed by the text "POSTED ON OCT 18, 2016 - [Edit this page on GitHub](#)". Below this is a paragraph of text: "Angular 2 presents two different methods for creating forms, template-driven (what we were used to in Angular 1.x), or reactive. We're going to explore the absolute fundamentals of the template-driven Angular 2 forms, covering `ngForm`, `ngModel`, `ngModelGroup`, submit events, validation and error messages."

<https://toddmotto.com/angular-2-forms-template-driven>



Model Driven Forms

Or: *Reactive Forms*



Reactive Forms

- Based on *reactive programming* we already know
 - Events, Event Emitters
 - Observables
- Every form control is an observable!

```
export abstract class AbstractControl {  
  
    ...  
    private _valueChanges: EventEmitter<any>;  
    ...  
    get valueChanges(): Observable<any> {  
        return this._valueChanges;  
    }  
    ...  
}
```



Differences - key things to remember

- No more `ngForm` → use `[formGroup]`
- No more `ngModel` → use `formControlName`
- Form state lives in the Component, *not* in the View
- Possible validations are in the Component, not in the View
- The view is *not* generated for you.
- You need to write the HTML yourself



Form Controls are observables

- Import & instantiate in the Component
- Build your model in constructor or `ngOnInit`.
- Listen to changes (`.subscribe()`) and act accordingly:

```
export class AppComponent1 implements OnInit {
```

```
  myReactiveForm: FormGroup;
```



```
  constructor(private FormBuilder: FormBuilder) {
```



```
  ngOnInit() {
```

```
    this.myReactiveForm = this.formBuilder.group({
```



```
      email    : '',
```

```
      password: ''
```

```
    })
```

```
  }
```

```
}
```



Subscribe to those observables

// 1. complete form

```
this.myReactiveForm.valueChanges.subscribe((value)=>{  
    console.log(value);  
});
```

// 2. watch just one control

```
this.myReactiveForm.get('email').valueChanges.subscribe((value)=>{  
    console.log(value);  
});
```




Building reactive forms



Step 1 – import ReactiveFormsModule

- `app.module.ts`

```
import {NgModule}      from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
import ...

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



Step 2 – use [formGroup] and formControlName

```
<form novalidate [formGroup]="myReactiveForm">
  <div class="form-group">
    <label for="inputEmail">Email address</label>
    <input type="email" class="form-control" id="inputEmail"
      placeholder="Enter email" name="email"
      formControlName="email">
  </div>
  ...
  // all other controls
</form>
```



Step 3 – Build your form in Component

```
export class AppComponent1 implements OnInit {  
  myReactiveForm: FormGroup;  
  constructor(private FormBuilder: FormBuilder) {  
  }  
  ngOnInit() {  
    // 1. Define the model of Reactive Form.  
    // Notice the nested FormBuilder.group() for group Customer  
    this.myReactiveForm = this.FormBuilder.group({  
      email    : ``,  
      password: ``,  
      customer: this.FormBuilder.group({  
        prefix: ``,  
        firstName: ``,  
        lastName: ``  
      })  
    })  
  }  
}
```



Subscribe to changes

```
ngOnInit() {
```

```
  ...
```

```
  // 2. Subscribe to changes at form level or...
```

```
  this.myReactiveForm.valueChanges.subscribe((value)=>{  
    console.log('Changes at form level: ', value);  
  });
```

```
  // 3. Subscribe to changes at control level.
```

```
  this.myReactiveForm.get('email').valueChanges.subscribe((value)=>{  
    console.log('Changes at control level: ', value);  
  });  
}
```



Submitting a reactive form

- Can be based on `.valueChanges()` (though not very likely) for any given form control or complete form
- Use just `.click()` event handler for submit button

```
<button type="submit" class="btn btn-primary"
  (click)="onSubmit()"
  [disabled]="!myReactiveForm.valid">
  Submit
</button>
```

```
onSubmit() {
  console.log('Form submitted: ', this.myReactiveForm.value);
  // TODO: do something useful with form
}
```



Form Validation



1. Validating Template driven forms

Use HTML5-attributes like `required`, `pattern`, `minlength` and so on.

Under the hood, these are actually Angular directives!

Angular adds/removes corresponding classes.



```
<input type="password" class="form-control" ngModel  
      id="inputPassword" placeholder="Password" name="password"  
      #pw="ngModel" required minlength="6">
```



Validating reactive forms

No more declarative attributes `required`, `minlength`, `maxlength` and so on.

Add `Validator` on the component class instead.

Configure validator per your needs.



Angular 2 built-in validators

[angular/modules/@angular/forms/src/validators.ts](#)

```
export class Validators {  
  
    static required(control: AbstractControl): {[key: string]: boolean} {  
    }  
  
    static minLength(minLength: number): ValidatorFn {  
    }  
  
    static maxLength(maxLength: number): ValidatorFn {  
    }  
  
    static pattern(pattern: string): ValidatorFn {  
    }  
  
    static nullValidator(c: AbstractControl): {  
    }  
  
    . . .  
}
```



Adding default Validators

- Adding Validators to class definition
 - `email : ['', Validators.required],`
- Multiple validations? Add an array of Validators, using `Validators.compose()`

```
this.myReactiveForm = this.formBuilder.group({  
  email : ['', Validators.required],  
  password: ['', Validators.compose([Validators.required, Validators.minLength(6)])],  
  confirm: ['', Validators.compose([Validators.required, Validators.minLength(6)])],  
  ...  
});
```



Adding Custom Validators

- Creating a Password-confirm validator
- Steps:
 1. Create a validation function, taking `AbstractControl` as a parameter
 2. Write your logic
 3. Don't forget: pass the function in as a configuration parameter for the group or form you are validating!



```
function passwordMatcher(control: AbstractControl) {  
    return control.get('password').value === control.get('confirm').value  
        ? null : {'nomatch': true};  
      
    // we *could* return just true/false here, but by returning an object  
    // we're more flexible in composing our validators.  
}
```



```
this.myReactiveForm = this.formBuilder.group({  
    email    : ['', Validators.required],  
    password: ['', Validators.compose([Validators.required, Validators.minLength(6)])],  
    confirm  : ['', Validators.compose([Validators.required, Validators.minLength(6)])],  
    },  
    {validator: passwordMatcher} // pass in the validator function  
);
```



More on FormBuilder class

- <https://angular.io/docs/ts/latest/api/forms/index/FormBuilder-class.html>
- Information on using and configuring FormBuilder

FormBuilder STABLE
CLASS

What it does

Creates an `AbstractControl` from a user-specified configuration.

It is essentially syntactic sugar that shortens the `new FormGroup()`, `new FormControl()`, and `new FormArray()` boilerplate that can build up in larger forms.

How to use

To use, inject `FormBuilder` into your component class. You can then call its methods directly.

```
1. import {Component, Inject} from '@angular/core';
2. import {FormBuilder, FormGroup, validators} from '@angular/forms';
3.
4. @Component({
5.   selector: 'example-app',
6.   template: `
```



Subscribing to form events

Working with Observables (again). Typeahead demo



Define a form

```
<form novalidate [formGroup]="searchForm">
  <div class="form-group">
    <label for="searchYouTube">Search YouTube</label>
    <input type="text" class="form-control" id="searchYouTube"
      formControlName="searchYouTube"
      placeholder="Search YouTube" name="search">
  </div>
</form>
```



Define component

- Compose a class, subscribe to `.valueChanges()` event

```
import {Http, Response} from '@angular/http';
import {Observable} from 'rxjs/Observable'
import {FormControl, FormGroup} from "@angular/forms";
...
// import just the operators we need, not import 'rxjs/Rx'
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/switchMap';
import 'rxjs/add/operator/debounceTime';

// define some constants
const BASE_URL = 'https://www.googleapis.com/youtube/v3/search';
const API_KEY = 'AIzaSyBdi3LXzf1xWXOAVgAwNkGvjnM1TwSV4VU';
// compose a url to search for, based on a query/keyword
const makeURL = (query: string) => `${BASE_URL}?q=${query}&part=snippet&key=${API_KEY}`;
```

```

@Component({
  selector    : 'component1',
  templateUrl: 'app/component1/app.component1.html'
})

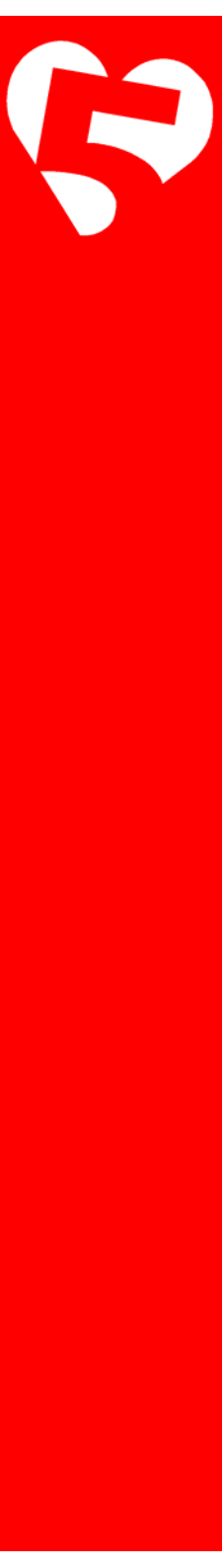
export class AppComponent1 implements OnInit {
  videos: Observable<any[]>;

  // compose our form
  searchYouTube = new FormControl();
  searchForm    = new FormGroup({
    searchYouTube: this.searchYouTube,
  });


  constructor(private http: Http) {
  }

  ngOnInit() {
    // subscribe to Youtube input textbox and bind async (see html)
    this.videos = this.searchYouTube.valueChanges
      .debounceTime(600)           // wait for 600ms to hit the API
      .map(query => makeURL(query)) // turn keyword into a real youtube-URL
      .switchMap(url => this.http.get(url)) // wait for, and switch to the Observable that my http get call returns (more
info on switchMap, for example https://egghead.io/lessons/rxjs-starting-a-stream-with-switchmap)
      .map((res: Response) => res.json()) // map its response to json
      .map(response => response.items);   // unwrap the response and return only the items array
  }
}



```




More on Reactive Forms



I'm Todd, a Developer Advocate @Telerik. Co-Founder of @UltimateAngular Creator of the Angular 2 migration guide. JavaScript, Angular, React, conference speaker. Developer Expert at Google.

 Follow @toddmotto  29.5K followers

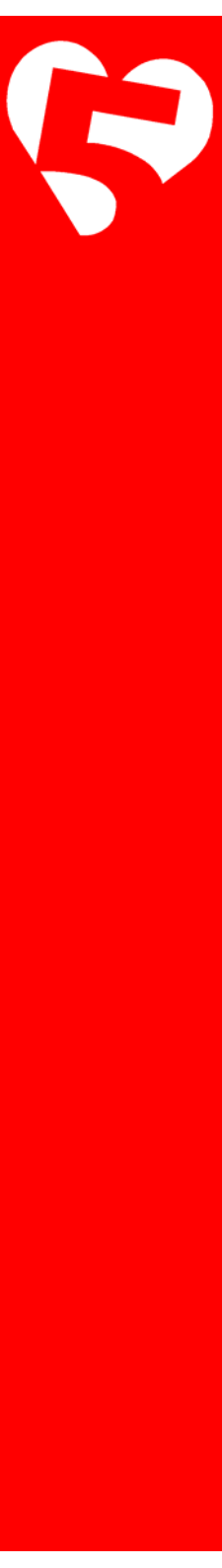
[Posts](#) [About](#) [Speaking](#) [Styleguide](#) [ngMigrate](#)  [Online training](#)

Angular 2 form fundamentals: reactive forms

POSTED ON OCT 19, 2016 - [Edit this page on GitHub](#)

Angular 2 presents two different methods for creating forms, template-driven (what we were used to in Angular 1.x), or reactive. We're going to explore the absolute fundamentals of the reactive Angular 2 forms, covering `ngForm` , `ngModel` , `ngModelGroup` , submit events, validation and error messages.

<https://toddmotto.com/angular-2-forms-reactive>



Kara Erickson on Angular Forms

FormsModule

Implicit creation of FormControl()
by directives

Source of truth: template



Async

ReactiveFormsModule


Explicit creation of FormControl()

Source of truth: component class


Sync



Angular 2 Forms | Kara Erickson

 AngularConnect

✓ Geabonneerd



8.523

7.965 weergaven

<https://www.youtube.com/watch?v=xYv9lsrV0s4>