Supercharge your

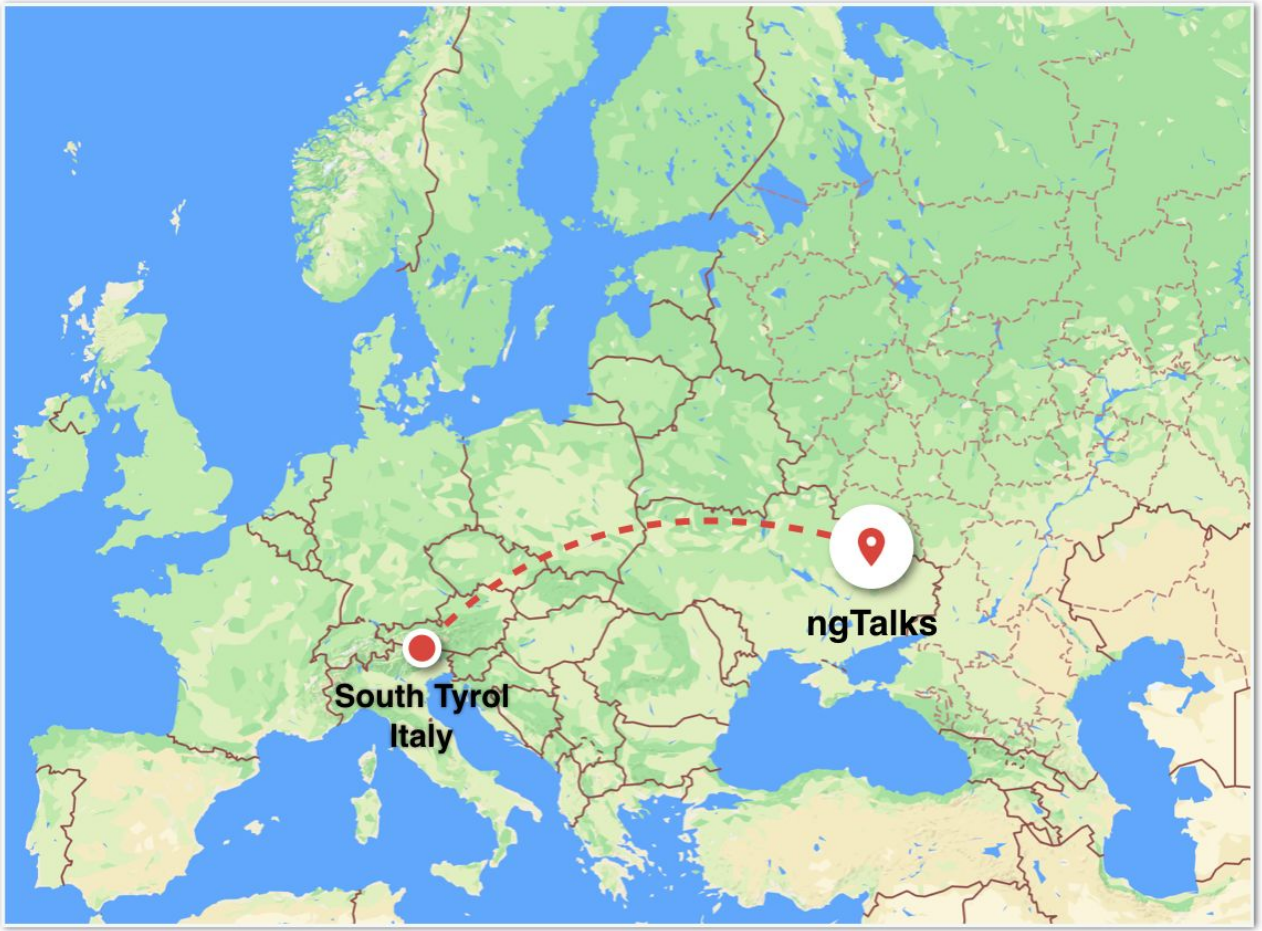
ANGULAR FORMS

@juristr • https://juristr.com

whoami?

JURI
STRUMPFLOHNER

Google Developers
Experts
Web Technologies



roterrucksack stefan_mahl vinumhotels

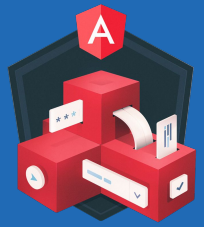
JURI STRUMPFLOHNER

R3-GIS
(r3-gis.com)

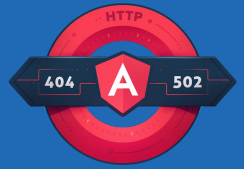
Trainer & Consultant

Egghead.io Instructor

Angular Service Injection with DI



Dynamic Forms



HTTP in Angular

Dynamic Components

egghead.io

Style Angular Components

JURI STRUMPFLOHNER

https://juristr.com

@juristr

github.com/juristr

Template driven forms

Model-driven forms

Template driven
forms

Reactive
forms

Template-driven Forms

@juristr

```
<form #form="ngForm">
   <input matInput placeholder="Name" [(ngModel)]="hero.name" name="name">
   <input matInput placeholder="Age" [(ngModel)]="hero.name" name="age">
   <mat-select placeholder="Superpower"
         [(ngModel)]="hero.superpower" name="superpower">
     <mat-option *ngFor="let power of powers" [value]="power">
         {{ power }}
     </mat-option>
   </mat-select>
   ...
</form>
```

@juristr

Template-driven Forms

(with some reeeeealllyyy simple validation)

```
<form #form="ngForm" (ngSubmit)="onSubmit(form)">
   <input matInput placeholder="Name" [(ngModel)]="hero.name"
         name="name" #nameInput="ngModel" required>
   <mat-error *ngIf="nameInput.invalid && nameInput.hasError('required')">
     Name is required
   </mat-error>
   ...
</form>
```

@juristr

Template-driven Forms

(with some reeeeeallyyy simple validation)

```
<form #form="ngForm" (ngSubmit)="onSubmit(form)">
   <input matInput placeholder="Name" [(ngModel)]="hero.name"
         name="name" #nameInput="ngModel" required>
   <mat-error *ngIf="nameInput.invalid && nameInput.hasError('required')">
     Name is required
   </mat-error>
   ...
</form>
```

“ You should use reactive forms !

! Reactive forms are more powerful

Reactive Forms

@juristr

```
export class HeroFormReactiveComponent implements OnInit {
 form: FormGroup;

 constructor(private fb: FormBuilder) {
   this.form = fb.group({
     name: ['', [Validators.required]],
     realName: '',
     superpower: ''
   });
 }

 ngOnInit() {
   this.form.patchValue(this.hero);
 }
}
```

Reactive Forms

@juristr

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
   <input matInput placeholder="Name" formControlName="name">
   <mat-error *ngIf="form.get('name').invalid && form.get('name').hasError('required')">
      Name is required</mat-error>
   </mat-form-field>

   <input matInput placeholder="Real Name" formControlName="realName">

   <mat-select placeholder="Superpower" formControlName="superpower">
     <mat-option *ngFor="let power of powers" [value]="power">{{ power }}</mat-option>
   </mat-select>
  ...
</form>
```

You Retweeted

Juri Strumpflohner
@juristr

#angular I'm curious, why would you choose reactive forms over template drive forms (or vice versa). What's the decisive factor for your choice? plz comment, thx 😍 #forms #angular

Reactive forms!!	**80%**
template driven!!!	20%

714 votes · Final results

9:17 PM · Jul 24, 2018

https://mobile.twitter.com/juristr/status/1021836886753591296

Meligy A ng-sydney
@Meligy

Replying to @juristr

Reactive forms all the way. Less magic, everything is in clear APIs

6:32 AM · Jul 25, 2018

https://mobile.twitter.com/Meligy/status/1021976444245864450

Michael Szczepaniak
@frontend_mike

Replying to @juristr

Complex business validations, tackling it in template driven forms would be a nightmare.

8:31 PM · Jul 25, 2018

https://mobile.twitter.com/frontend_mike/status/1022187601304997888

faz [Ξ]
@_faz

Replying to @juristr

reactive forms easier to test, in my opinion

12:05 AM · Jul 26, 2018

https://mobile.twitter.com/_faz/status/1022241576179957760

John Papa
@John_Papa

Replying to @juristr

I use both. Template forms in pretty much every tech has always felt a bit too heavy in the markup/UI world. I feel better using code to handle form interactions.

That said, reactive forms are far more than using code to handle forms. It's also reactive (thus the name).

1:04 AM · Jul 25, 2018

https://mobile.twitter.com/John_Papa/status/1021894018240835585

John Papa
@John_Papa

Replying to @John_Papa and @juristr

I believe reactive forms are hard to learn and implement for new folks as they have to learn reactive too (yet another concept). Would be interesting to see a code based model driven forms approach.

1:05 AM · Jul 25, 2018

https://mobile.twitter.com/John_Papa/status/1021894018240835585

Olivier Combe
@OCombe

Replying to @juristr

I use both, depending on how complex the form is (simple = template)

9:47 PM · Jul 24, 2018

https://mobile.twitter.com/OCombe/status/1021844393450463232

Aleš
@fxck01

Replying to @juristr

I wish I wouldn't have to use either. Easily my least favorite angular package.

7:47 AM · Jul 27, 2018

https://mobile.twitter.com/fxck01/status/1022720176125562880

Lots of **typing**!

Multiple places

Reactive Forms - **Duplicate declarations**

@juristr

```
export class HeroFormReactiveComponent implements OnInit {
 form: FormGroup;

 constructor(private fb: FormBuilder) {
   this.form = fb.group({
     name: ['', [Validators.required]],
     realName: '',
     superpower: ''
   });
 }

 ngOnInit() {
   this.form.patchValue(this.her
 }
}
```

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
   <input matInput placeholder="Name" formControlName="name">
   <mat-error *ngIf="form.get('name') invalid && form.get('name').hasError(
      Name is required</mat-error>
   </mat-form-field>

   <input matInput placeholder="Real Name" formControlName="realName">

   <mat-select placeholder="Superpower" formControlName="superpower">
```

Dynamic Forms

?

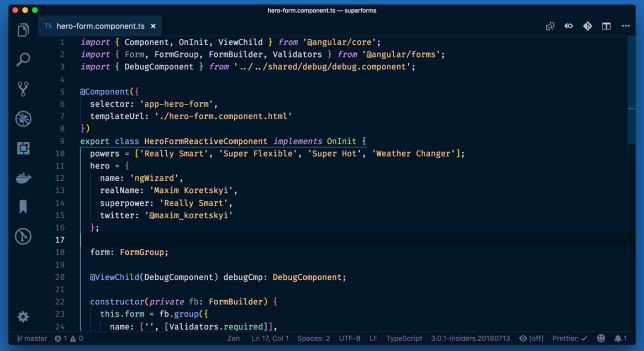
Levels

of “dynamicity”

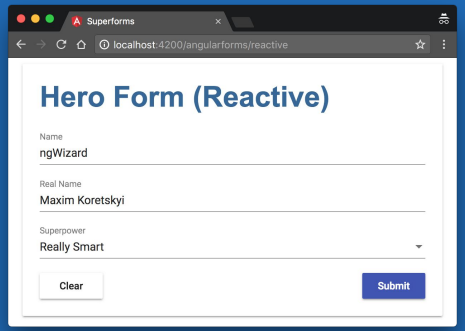
1. We write the configuration in code
   HTML is generated

2. User defines the data model
   Configuration + HTML is generated

Meaning...



Change TypeScript code



HTML automatically updates

Simple, homemade, dyn form

```
@Component({
 selector: 'app-dynamic-form',
 template: `
   <form [formGroup]="form">
     <input type="text" formControlName="firstname">
   </form>
 `
})
export class DynamicFormComponent implements OnInit {
   ...
}
```

Simple, homemade, dyn form

@juristr

```
@Component({ … })
export class DynamicFormComponent implements OnInit {
 form: FormGroup;
 firstnameValue = 'Juri';

 ngOnInit() {
   this.form = new FormGroup({
     firstname: new FormControl(this.firstnameValue)
   });
 }
}
```

Simple, homemade, dyn form

@juristr

```
@Component({
 template: `
   <form [formGroup]="form">
     <input type="text" [formControlName]="fieldName">
   </form>
 `
})
export class DynamicFormComponent implements OnInit {
 fieldName = 'firstname';

 ngOnInit() {
   this.form = new FormGroup({
     [this.fieldName]: new FormControl(this.firstnameValue)
   });
```

Let’s do this for an entire person object:

```
person = {
 firstname: 'Juri',
 age: 33,
 surname: 'Strumpflohner',
 twitter: '@juristr'
};
```

Generate the FormGroup...

```
this.form = new FormGroup({
    firstname: new FormControl(''),
    age: new FormControl(''),
    ...
});
```

Generate the FormGroup...

```
const formDataObj = Object.keys(this.person).reduce((formObj, prop) => {
   formObj[prop] = new FormControl(this.person[prop]);
   return formObj;
}, {});

this.form = new FormGroup(formDataObj);
```

Generate the FormGroup...

```
const formDataObj = Object.keys(this.person) reduce((formObj, prop) => {
    formObj[prop] = new FormControl(this.person[prop]);
    return formObj;
}, {});

this.form = new FormGroup(formDataObj);
```

Generate the FormGroup...

```
const formDataObj = Object.keys(this.person).reduce((formObj, prop) => {
    formObj[prop] = new FormControl(this.person[prop]);
    return formObj;
}, {});

this.form = new FormGroup(formDataObj);
```

Generate the FormGroup...

```
const formDataObj = Object.keys(this.person).reduce((formObj, prop) => {
  formObj[prop] = new FormControl(this.person[prop]);
  return formObj;
}, {});

this.form = new FormGroup(formDataObj);
```

Generate the FormGroup...

```
const formDataObj = Object.keys(this.person).reduce((formObj, prop) => {
    formObj[prop] = new FormControl(this.person[prop]);
    return formObj;
}, {});

this.form = new FormGroup(formDataObj);
```

...and finally the template

```
<form [formGroup]="form">
   <mat-form-field class="full-width" *ngFor="let prop of personProps">
     <input matInput type="text" [placeholder]="prop" [formControlName]="prop">
   </mat-form-field>
</form>
```

...and finally the template

```
<form [formGroup]="form">
   <mat-form-field class="full-width" *ngFor="let prop of personProps">
     <input matInput type="text" [placeholder]="prop" [formControlName]="prop">
   </mat-form-field>
</form>
```

...and finally the template

```
<form [formGroup]="form">
   <mat-form-field class="full-width" *ngFor="let prop of personProps">
     <input matInput type="text" [placeholder]="prop" [formControlName]="prop">
   </mat-form-field>
</form>
```

Tada 🎉

@juristr

firstname

Juri

age

33

surname

Strumpflohner

twitter

@juristr

website

https://juristr.com

Tada 🎉

@juristr

Debug

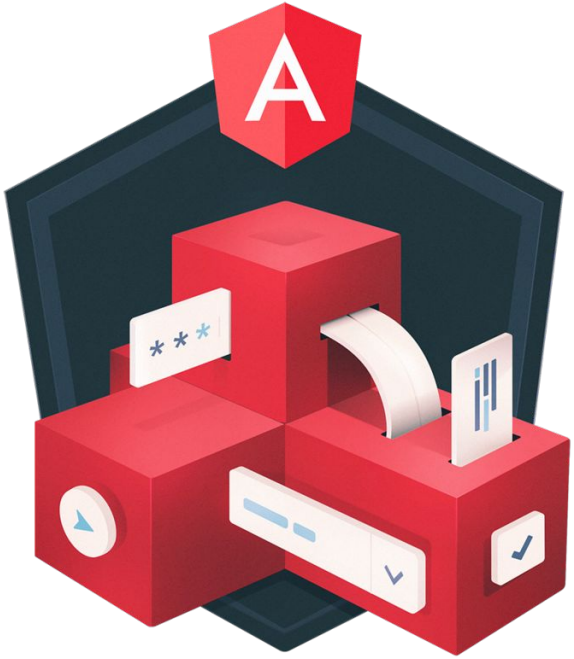
Form value | Model | Submitted


```
{
  "firstname": "Juri",
  "age": "33",
  "surname": "Strumpflohner",
  "twitter": "@juristr",
  "website": "https://juristr.com"
}
```

What about the input type?

Well...bind it as well!

```
<form [formGroup]="form">
   <mat-form-field class="full-width" *ngFor="let prop of personProps">
     <input matInput [type]="prop.type" [formControlName]="prop.name">
   </mat-form-field>
</form>
```



Create Dynamic Forms in Angular

(https://egghead.io/courses/create-dynamic-forms-in-angular)

What about form validation?

What about ...?

...?

Field config

&

Model

Remember **angular-formly**?

(psst, AngularJS 1.x)

FORMLY Guides UI Examples GitHub

Angular Formly

Easy Reactive Forms in Angular

Get started

formly-js/ngx-formly

https://formly-js.github.io/ngx-formly/

Angular Material @ngx-formly/material

Bootstrap @ngx-formly/bootstrap

PrimeNG @ngx-formly/primeng

Kendo-UI @ngx-formly/kendo

Ionic @ngx-formly/ionic

NativeScript @ngx-formly/nativescript

@ngx-formly/core

Installation

Installing **ngx-formly**

```
$ npm install @ngx-formly/core
$ npm install @ngx-formly/material


// or
$ yarn add @ngx-formly/core
$ yarn add @ngx-formly/material
```

ng add **<magic>** powered by **@angular-devkit/schematics**

```
$ ng add @ngx-formly/schematics --ui-theme=material
```

Ready to go!

```
@NgModule({
 declarations: [AppComponent],
 imports: [
   ...
   ReactiveFormsModule,
   FormlyModule.forRoot(),
   FormlyMaterialModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule {}
```

Formly

Basics

Firstname

Juri

Surname

Strumpflohner

Twitter

@juristr

Clear

Submit

Field config

&

Model

Formly Field Configuration
(`FormlyFieldConfig`)

configs the form

defines "type" of fields

property bindings

Validation

HTML control properties
(visibility, dependencies, lifecycle)

Formly field configuration

```
fields: FormlyFieldConfig[] = [
    {
      key: 'firstname',
      type: 'input',
      templateOptions: {
         label: 'Firstname'
      }
    }
]
```

Model?

data object

want to modify

send back to our data store

Model

```
model: Person = {
  id: 1223,
  firstname: 'Juri',
  twitter: '@juristr'
};
```

Configuring the component template

Component **Template**

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
 <formly-form [model]="model" [fields]="fields" [form]="form">
   <div>
     <button type="reset" mat-raised-button>Clear</button>
     <button mat-raised-button color="primary">Submit</button>
   </div>
 </formly-form>
</form>
```

Component **Template**

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
  <formly-form [model]="model" [fields]="fields" [form]="form">
    <div>
      <button type="reset" mat-raised-button>Clear</button>
      <button mat-raised-button color="primary">Submit</button>
    </div>
  </formly-form>
</form>
```

Component **Template**

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
 <formly-form [model]="model" [fields]="fields" [form]="form">
   <div>
     <button type="reset" mat-raised-button>Clear</button>
     <button mat-raised-button color="primary">Submit</button>
   </div>
 </formly-form>
</form>
```

Component **Template**

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
 <formly-form [model]="model" [fields]="fields" [form]="form">
   <div>
     <button type="reset" mat-raised-button>Clear</button>
     <button mat-raised-button color="primary">Submit</button>
   </div>
 </formly-form>
</form>
```

Component **Template**

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
 <formly-form [model]="model" [fields]="fields" [form]="form">
   <div>
     <button type="reset" mat-raised-button>Clear</button>
     <button mat-raised-button color="primary">Submit</button>
   </div>
 </formly-form>
</form>
```

Adjust
input type

input type **number**

age

33

```
{
  key: 'age',
  type: 'input',
  templateOptions: {
    label: 'Age',
    type: 'number'
  }
}
```

input type **select**

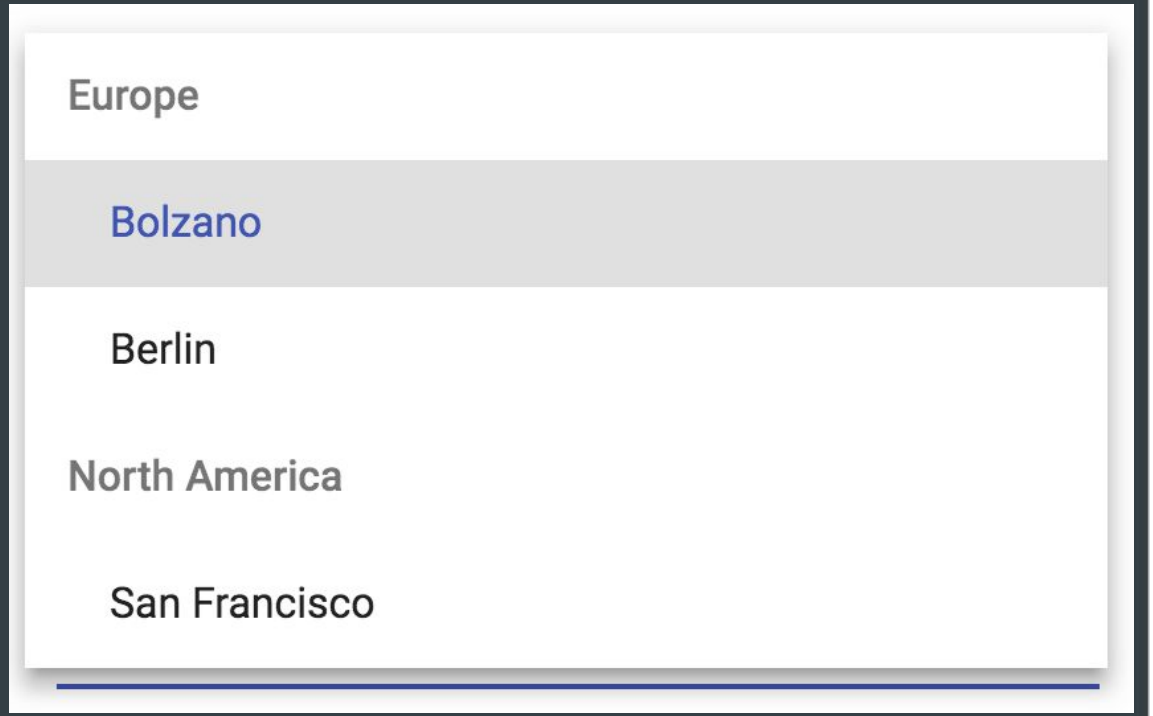
city

Bolzano

```
{
 key: 'cityId',
 type: 'select',
 templateOptions: {
   label: 'city',
   options: [
     {
       value: 1,
       label: 'Bolzano'
     },
     ...
   ]
```

input type **select** (grouping)

```
templateOptions: {
 label: 'city',
 options: [
   {
     value: 1,
     label: 'Bolzano',
     group: 'Europe'
   }
 ],
 groupProp: 'group',
 valueProp: 'value',
 labelProp: 'label',
}
```



Observable support

select **with observables**

```
import { of } from 'rxjs';
...
cities$ = of([
 {
   value: 1,
   label: 'Bolzano',
   group: 'Europe'
 },
 {
   value: 2,
   label: 'Berlin',
   group: 'Europe'
 },
 ...
]);
```

select **with observables**

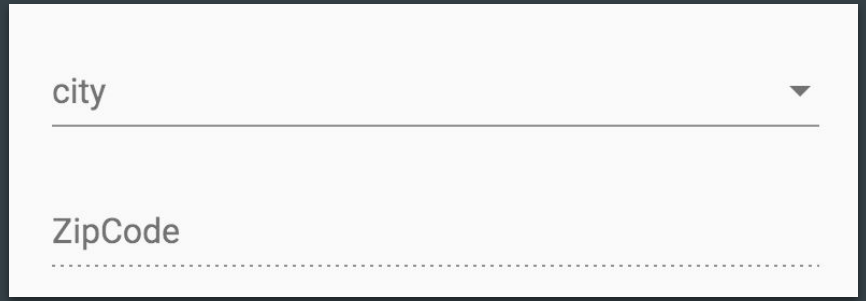
```
{
 key: 'cityId',
 type: 'select',
 templateOptions: {
   label: 'city',
   options: this.cities$,
   groupProp: 'group'
 }
}
```

Expressions

Visibility, disabled/enabled,...

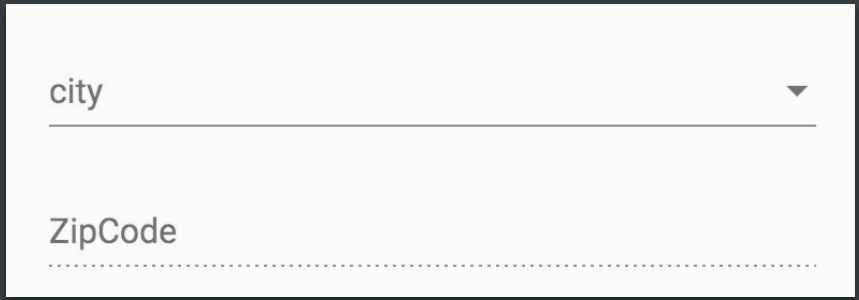
enable/disable based on another field

```
{
 key: 'zipCode',
 type: 'input',
 templateOptions: {
   label: 'ZipCode'
 },
 expressionProperties: {
   'templateOptions.disabled': '!model.cityId'
 }
}
```



enable/disable based on another field

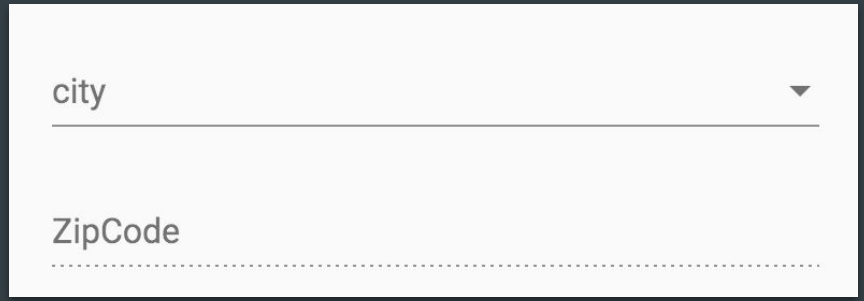
```
{
 key: 'zipCode',
 type: 'input',
 templateOptions: {
   label: 'ZipCode'
 },
 expressionProperties: {
   'templateOptions.disabled': (model, formState) => !model.cityId
 }
}
```



@juristr

Psst! v5 will have **observable expressions**

```
{
 key: 'zipCode',
 type: 'input',
 templateOptions: {
   label: 'ZipCode'
 },
 expressionProperties: {
   'templateOptions.disabled': this.someObservable$
 }
}
```



Hide expressions

```
{
 template: `<p><strong>Awesome</strong>, thanks for choosing your city!</p>`,
 hideExpression: model => !model.cityId
}
```

city

Bolzano

ZipCode

Awesome, thanks for choosing your city!

Input validation

Name *

This field is required

age

15

Sorry, you have to be of legal age.

city

Clear

Submit

Formly got you covered as well

Basic validators (required, min, max,...)

```
{
 key: 'name',
 type: 'input',
 templateOptions: {
   label: 'Name',
   required: true
 }
}
```

Validation Messages - Globally

```
@NgModule({
 ...
 imports: [
   FormlyModule.forRoot({
     validationMessages: [
       { name: 'required', message: 'This field is required' }
     ]
   }),
   ...
 ]
})
export class AppModule {}
```

Validation Messages - Locally

```
{
  ...
  templateOptions: {
    label: 'age',
    type: 'number',
    min: 18
  },
  validation: {
    messages: {
      min: 'Sorry, you have to be of legal age.'
    }
  }
}
```

Conditional required validation? Here ya go!

```
{
 key: 'zipCode',
 type: 'input',
 templateOptions: {
   label: 'Zip Code'
 },
 expressionProperties: {
   'templateOptions.required': model => !!model.cityId
 }
}
```

Show me some Advanced stuff !

Dependent Fields

Nation

Italy

city

Rome

Clear

Submit

Lifecycle hooks

```
lifecycle {

}
},
{
ke
ty
te
},
```

- afterContentChecked?
- afterContentInit?
- afterViewChecked?
- afterViewInit?
- doCheck?
- onChanges?
- onDestroy?
- onInit?

Lifecycle hooks

```
{
 key: 'cityId',
 type: 'select',
 templateOptions: {
   label: 'city'
 },
 lifecycle: {
   onInit: (form, field, model) => {
      ...
   }
}
```

Lifecycle hooks

@juristr

```
{
 ...
 lifecycle: {
   onInit: (form, field, model) => {
     form.get('nationId').valueChanges.pipe(
         tap(nationId => {
           field.formControl.setValue(null);
           field.templateOptions.options = this.cities$.pipe(
             map(cities => cities.filter(x => x.nationId === nationId))
           );
         }),
         takeUntil(this.destroy$)
       ).subscribe();
```

Lifecycle hooks

@juristr

```
{
 ...
 lifecycle: {
   onInit: (form, field, model) => {
     form.get('nationId').valueChanges.pipe(
         tap(nationId => {
           field.formControl.setValue(null);
           field.templateOptions.options = this.cities$.pipe(
             map(cities => cities.filter(x => x.nationId === nationId))
           );
         }),
         takeUntil(this.destroy$)
       ).subscribe();
```

Lifecycle hooks

@juristr

```
{
 ...
 lifecycle: {
   onInit: (form, field, model) => {
     form.get('nationId').valueChanges.pipe(
         tap(nationId => {
           field.formControl.setValue(null);
           field.templateOptions.options = this.cities$.pipe(
             map(cities => cities.filter(x => x.nationId === nationId))
           );
         }),
         takeUntil(this.destroy$)
       ).subscribe();
```

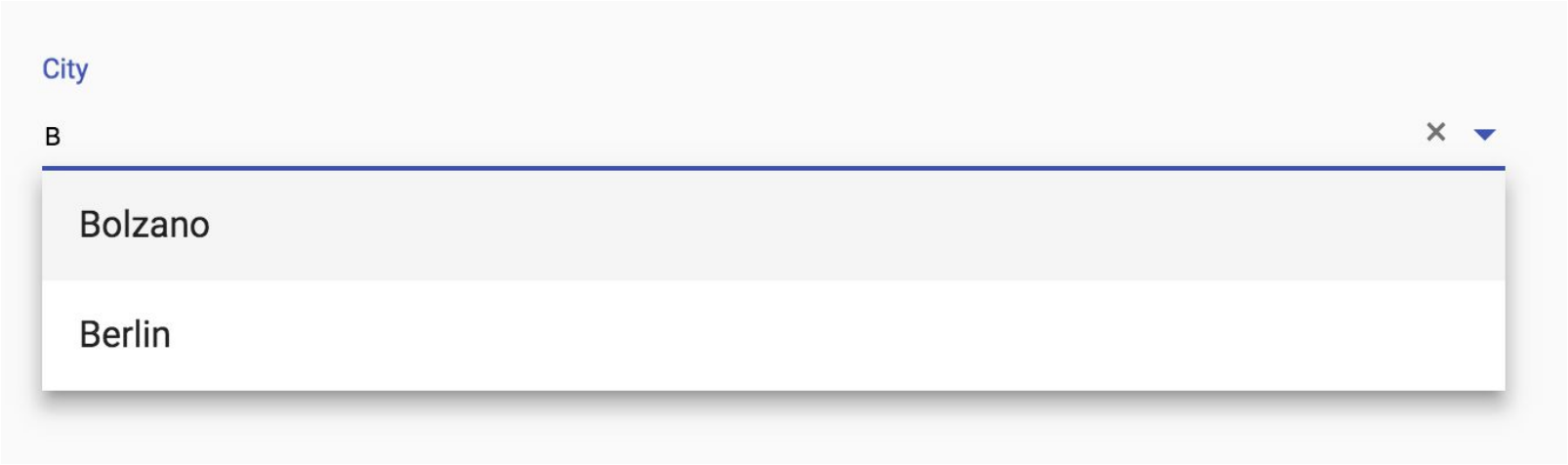
Lifecycle hooks

@juristr

```
{
 ...
 lifecycle: {
   onInit: (form, field, model) => {
     form.get('nationId').valueChanges.pipe(
         tap(nationId => {
           field.formControl.setValue(null);
           field.templateOptions.options = this.cities$.pipe(
             map(cities => cities.filter(x => x.nationId === nationId))
           );
         }),
         takeUntil(this.destroy$)
       ).subscribe();
```

Custom FormControls

extend ngx-formly



https://github.com/ng-select/ng-select

Create a **Formly Type**

Works as a wrapper around components

```
@Component({
 selector: 'formly-ng-select',
 template: `
   <div class="mat-input-infix mat-form-field-infix">
      <ng-select [items]="to.options | async"...>
      </ng-select>
   </div>
 `
})
export class NgSelectFormlyComponent extends FieldType {}
```

Create a **Formly Type**

Works as a wrapper around components

```
@Component({
 selector: 'formly-ng-select',
 template: `
   <div class="mat-input-infix mat-form-field-infix">
      <ng-select [items]="to.options | async"...>
      </ng-select>
   </div>
 `
})
export class NgSelectFormlyComponent extends FieldType {}
```

Create a **Formly Type**

```
<div class="mat-input-infix mat-form-field-infix">
  <ng-select [items]="to.options | async"
     [placeholder]="to.label"
     [bindValue]="to.bindValue || 'value'"
     [formControl]="formControl"
     [class.is-invalid]="showError">
 </ng-select>
</div>

export class NgSelectFormlyComponent extends FieldType {}
```

Create a **Formly Type**

```
<div class="mat-input-infix mat-form-field-infix">
  <ng-select [items]="to.options | async"
     [placeholder]="to.label"
     [bindValue]="to.bindValue || 'value'"
     [formControl]="formControl"
     [class.is-invalid]="showError">
 </ng-select>
</div>

export class NgSelectFormlyComponent extends FieldType {}
```

"to" = TemplateOptions

Inhertited from FieldType

Register the new Formly Type

```
FormlyModule.forRoot({
  ...
  types: [
    {
      name: 'my-autocomplete',
      component: NgSelectFormlyComponent
    }
  ]
})
```

Use it!

```
fields: FormlyFieldConfig[] = [
 {
   key: 'cityId',
   type: 'my-autocomplete',
   templateOptions: {
     label: 'City',
     options: this.cityService.getCities()
   }
 }
];
```

Read the source!

https://github.com/formly-js/ngx-formly/tree/master/src/material

Custom “DSL”

Reusability FTW!

Create a Formly Service

```
@Injectable(...)
export class FormlyService {
 input(config: R3FormlyFieldConfig): FormlyFieldConfig {}
 number(config: R3FormlyFieldConfig): FormlyFieldConfig {}
 select(config: R3FormlyFieldConfig): FormlyFieldConfig {}
 autocomplete(config: R3FormlyFieldConfig): FormlyFieldConfig {}
 ...
}
```

..then in your component

```
fields: FormlyFieldConfig[] = [
  this.formly.input({ key: 'firstname', label: 'Firstname' }),
  this.formly.input({ key: 'surname', label: 'Surname' }),
  this.formly.number({ key: 'age', label: 'Age' }),
  this.formly.select({
    key: 'cityId',
    label: 'City',
    data: this.cityService.getCities()
  })
];
```

You can even go a step further...

```
fields: FormlyFieldConfig[] = [
  this.formly.input({ key: 'firstname', label: 'Firstname' }),
  this.formly.input({ key: 'surname', label: 'Surname' }),
  this.formly.number({ key: 'age', label: 'Age' }),

  this.formly.citySelect()
];
```

- Avoid repetitive Formly config
- Reuse!
- Tailored to your app
- Easier handle (potentially) breaking changes on Formy

Repeats

Tabs Form

Forms with Datatable

Nested Forms

Multi-step

Flex based layouts

JSON powered

https://formly-js.github.io/ngx-formly/examples/introduction

Concluding

> When I'm doing a project it's not a question *if* I'll add Formly, the question is *when* I'll add it.

Bram Borggreve
@beeman_nl

Powerful!

- Forms are super easy and fast to write!
- highly customizable & flexible (custom formly types,..)
- lots of forms, heavy data-driven apps
- High reusability across your various forms
- when you truly have dynamic data models, defined by your user

Needs more work when...

- Lots of varying & very particular form layouts
- 3rd party UI controls (of a lib not supported by ngx-formly)
- Having a custom company style guide that is not based either on Material nor Bootstrap etc...

Thanks!

Abdellatif Ait boudad

@aitboudad

Core Maintainer of ngx-formly

JURI
STRUMPFLOHNER

Questions?

Reach out to me later &
connect with me on Twitter

@juristr