

Forms, validation & functions

Angular



Types of forms in Angular

- Databinding + formulieren was het USP van de eerste versies van Angular
- Template driven forms
- Model driven forms

Template driven forms

- Gebruiken ngModel bindings
 - 2 way binding
- Ondersteunen validatie vanuit de HTML
 - Angular validation / HTML5 validation
- End-to-end testing methodes
- Voornamelijk gebruikt bij formulieren van medium grootte waar weinig interactie nodig is

Template driven forms

- Om template driven forms te gebruiken, moet de FormsModule toegevoegd worden aan de app.module.ts file:

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

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

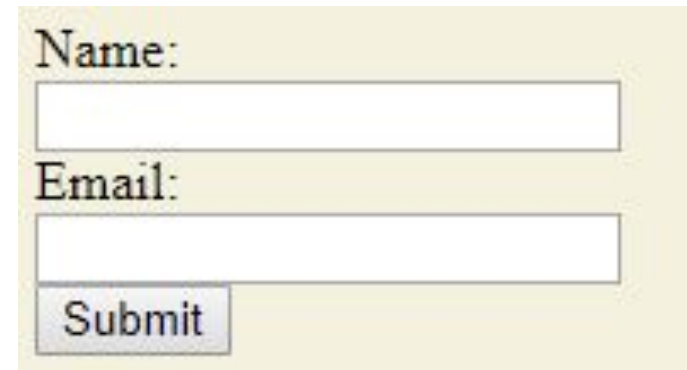
Template driven forms - voorbeeld

- In volgend voorbeeld gebruiken we het form object
- Input velden worden gekoppeld aan het formulier
- Een functie wordt gekoppeld aan de submit button
- Data van de velden wordt meegegeven aan de submit button

Template driven forms - voorbeeld

- Starten van volgende code in een default component:

```
<form>  
  Name:<br/>  
  <input type="text" name="name" [(ngModel)]="name" /> <br/>  
  Email:<br/>  
  <input type="email" name="email" [(ngModel)]="email" /><br/>  
  <button type="submit">Submit</button>  
</form>
```



Template driven forms - voorbeeld

- De koppeling van de input velden gebeurt aan de hand van 2-way binding:
 - [(ngModel)]

```
<input type="text" name="name" [(ngModel)]="name" />
```

- De 2-way binding zorgt er ook voor dat de input velden binnen een <form> tag gekoppeld worden aan een form object.
- Dit form object bevat data over:
 - Input van de velden
 - Informatie over de validatie van het formulier

Template driven form - voorbeeld

- Het aangemaakte Angular form object kan je gebruiken door een referentie te voorzien als volgt:

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

- Variable 'mijnForm' wordt aangemaakt met het form object. Dit object bevat:
 - mijnForm.form.value: object met de waarden van input velden
 - mijnForm.form.valid: boolean of alle validatieregels valid zijn
 - mijnForm.form.submitted: boolean of het form gesubmit is.

Template driven form - voorbeeld

- Koppelen van het submitten van het formulier aan een methode in de klasse:

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

- Het object mijnForm wordt meegegeven in de methode
 - Op deze manier krijgt de methode toegang tot de values & status van het formulier
- Een element van het type submit zorgt voor het triggeren van de onSubmit methode
 - “action” attribuut is niet nodig!

Template driven form - voorbeeld

- De onSubmit methode wordt voorzien in de component klasse:

```
onSubmit(form) {  
  console.log('Submitted:' +  
    JSON.stringify(form.value, null, 2));  
}
```

- De form.value waardes zijn gekoppeld aan het “**name**” attribuut van de input velden.

Template driven form - voorbeeld

- Templateform.component.ts:

```
export class TemplateformComponent {  
  onSubmit(form) {  
    console.log('Submitted:' + JSON.stringify(form.value, null, 2));  
  }  
}
```

- Templateform.component.html:

```
<form #mijnForm="ngForm"  
(ngSubmit)="onSubmit(mijnForm)">  
  Name:<br/>  
  <input type="text" name="name" [(ngModel)]="name" /> <br/>  
  Email: <br/>  
  <input type="email" name="email" [(ngModel)]="email" /><br/>  
  <button type="submit">Submit</button>  
</form>
```

Template driven form - voorbeeld

Name:

Email:

Elements

Console

top

Filter

Default levels

Angular is running in the [core.es5.js:2925](#) development mode. Call `enableProdMode()` to enable the production mode.

Submitted:{ [templateform.component.ts:13](#)
 "name": "Dries Swinnen",
 "email": "dries.swinnen@pxl.be"
}

Template driven forms – HTML Validatie

- Standaard Angular validation actief
- Validatie kan aan de hand van HTML5 validatie:
 - HTML5 attribuut 'required'
 - Input types zoals date, email, url, ...
- Wil je gewone HTML5 validatie gebruiken moet je **ngNativeValidate**

```
<form #mijnForm="ngForm" ngNativeValidate  
(ngSubmit)="onSubmit(mijnForm)">  
Name:<br/>  
<input type="text" name="name" [(ngModel)]="name" required />
```

Template driven forms – Angular validatie

- Naast HTML5 validatie kan je ook Angular validatie gebruiken
 - Angular validatie staat standaard actief
- Meer mogelijkheden
- Betere integratie met je component & het Angular framework:
 - Error message na het verlaten van een inputveld (dynamisch)
 - Aangepaste error messages
 - Programmatorisch testen van form validatie
 - Zelf schrijven van validatieregels

Template driven forms – Angular validatie

- Angular heeft volgende validators die toegevoegd kunnen worden aan input elementen binnen template driven forms:
 - Required
 - minLength
 - maxLength
 - Pattern
- Combinatie van verschillende validators is mogelijk

Template driven forms – Angular validatie

```
<form>
  <input type="text" required name="name" [(ngModel)]="name">
  <input type="text" minlength="3" name="street" [(ngModel)]="street">
  <input type="text" maxlength="8" name="city" [(ngModel)]="city">
  <input type="text" pattern="[A-Za-z]{5}" name="zip" [(ngModel)]="zip">
</form>
```


Template driven forms – Angular validatie

- Angular validatie heeft een deel dynamische booleans die gebruikt kunnen worden om de status van het formulier te raadplegen:
 - Valid & invalid: voldoet aan validatie
 - Touched & untouched: is de gebruiker in het veld geweest
 - Dirty & pristine: heeft de gebruiker wijzigingen aangebracht aan het veld
- Deze booleans kunnen geraadpleegd worden via het form object:

```
<p>  
Form valid: {{mijnForm.form.valid}} <br/>  
Form touched: {{mijnForm.form.touched}}  
</p>
```

Template driven forms – Angular validatie

- De status kan ook geraadpleegd worden op veld niveau in het formulier.
 - Template variabele voorzien:

```
<input type="text" name="name" [(ngModel)]="name" required  
#firstname="ngModel"/> <br/>
```

- Template variabele raadplegen in view:

```
Name veld valid: {{firstname.valid}} <br/>  
Name veld touched: {{firstname.touched}} <br/>  
Name veld dirty: {{firstname.dirty}}
```

Template driven forms – Angular validatie

- Er worden dynamisch css classes gekoppeld aan velden op basis van de status. Onderstaande classes kunnen gebruikt worden:

Control status	Class als waar	Class als onwaar
Input veld is bezocht (focus)	ng-touched	ng-untouched
Waarde is gewijzigd	ng-dirty	ng-pristine
Waarde is 'valid'	ng-valid	ng-invalid

- CSS classes kunnen op basis van deze zaken aangepast worden:

```
input.ng-invalid{ border: 2px solid red}
input.ng-valid{ border: 2px solid green}
```

Template driven forms – Form submit

- `mijnForm.form.valid` kan ook gebruikt worden in combinatie met bv. `[disabled]`

```
<button type="submit"  
[disabled]="!mijnForm.form.valid">Submit</button>
```

- De submit knop wordt zichtbaar als de boolean `valid` `true` is
- Bij grote formulieren kan een klasse object gekoppeld worden aan een formulier voor een beter overzicht van data.

Template driven forms – Classes

```
// registration.class.ts
export class Registration {
  name: string;
  email: string;
}
```

```
// form2.component.ts
export class Form2Component {
  constructor(private reg: Registration) { }
  onSubmit(data){
    console.log("Submitted:" + JSON.stringify(reg));
  }
}
```

```
<!-- form2.component.html -->
<input type="text" name="name" [(ngModel)]="registration.name" />
<input type="email" name="email" [(ngModel)]="registration.email" />
```

Template driven forms – Input types

- Verschillende input types kunnen gebruikt worden:
 - Input velden:
 - Tekst
 - Email
 - Checkbox
 - ...
 - Dropdown menu's:

```
<select name="product" [(ngModel)]=“product”>  
<option *ngFor=“let item of products” [value]=“item”>{{item}}</option>  
</select>
```

```
//in component klasse:  
products: string[] = [“Laptop”, “Tablet”, “Phone”, “Watch”];
```

Template driven forms – Input types

- Verschillende input types kunnen gebruikt worden:

- Datum velden:

```
<input type="date" [(ngModel)] = "dateStr" name="dateStr"
```

- Radio buttons:

```
<input id="one" type="radio" value="one" name="choice"  
[(ngModel)]="choice"><label for="one">One</label>  
<input id="two" type="radio" value="two" name="choice"  
[(ngModel)]="choice"><label for="two">Two</label>  
<input id="three" type="radio" value="three" name="choice"  
[(ngModel)]="choice"><label for="three">Three</label>
```

Model driven forms

- Ook wel reactive forms genoemd
- Form control objecten worden gemanipuleerd vanuit de component klasse
- De vormgeving / validatie gebeurt niet meer volledig in de HTML, maar meer in de component klasse,
- De component klasse kan kijken naar veranderingen in het formulier om vervolgens erop te “reageren” (=reactive).

Model driven forms

- Om modeldriven forms te gebruiken, moet de ReactiveFormsModule toegevoegd worden aan de app.module.ts file:

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

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

Model driven forms - voorbeeld

- In volgend voorbeeld gebruiken we een FormGroup
- Input velden worden gekoppeld aan de FormGroup in de component klasse
- Een functie wordt gekoppeld aan de submit button
- Data van de velden wordt meegegeven aan de FormGroup

Model driven forms - voorbeeld

- In de component.ts file worden 3 imports voorzien:

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

- In de component klasse wordt een FormGroup object aangemaakt en gedefinieerd in de ngOnInit methode:
 - De namen van de FormControls worden gebruikt in het formulier
 - De FormControl krijgt 2 argumenten:
 - Een default value: waarde of null
 - Eén validator of een array met validators

Model driven forms - voorbeeld

```
export class ModelformComponent implements OnInit {  
  myForm: FormGroup;  
  
  ngOnInit(): void {  
    this.myForm = new FormGroup({  
      first: new FormControl('Dries', []),  
      last: new FormControl('Swinen', [Validators.required])  
    });  
  }  
}
```

Model driven forms - voorbeeld

- In de component.html file voorzien we het formulier met koppeling naar de FormGroup
- Elk element krijgt een verwijzing naar de FormControl in de FormGroup

```
<form [formGroup]="myForm">  
First name: <input FormControlName="first"> <br/>  
Last name: <input FormControlName="last"> <br/>  
<input type="button" value="Submit" (click)=onSubmit() />  
</form>
```

Model driven form - voorbeeld

- De onSubmit methode wordt voorzien in de component klasse:

```
onSubmit() {  
    console.log('Submitted: ' +  
        JSON.stringify(this.myForm.value, null, 2));  
}
```

- myForm verwijst naar de FormGroup. Je kan FormControl's ook afzonderlijk opvragen:

```
console.log('Submitted: ' + this.myForm.get('first').value);  
console.log('Submitted: ' + this.myForm.get('last').value);
```

Model driven form - validatie

- Validatie wordt niet in de HTML code voorzien maar in de declaratie van de FormControl objecten
- Eén of meerdere validators kunnen gekoppeld worden
- Volgende build-in validators zijn beschikbaar:
 - required
 - minLength
 - maxLength
 - Pattern
- Zelf validators schrijven is mogelijk

Model driven form - validatie

- Validators worden toegevoegd aan de FormControl:

```
this.myForm = new FormGroup({  
  first: new FormControl('Dries', [Validators.required,  
Validators.minLength(3)]),  
  last: new FormControl('Swinnen',  
[Validators.pattern('[A-Z][a-z]*')]),  
});
```

- Pattern kan zowel een regex zijn of een string met een regex (zie voorbeeld lab)
- Referentie in de view:

```
<input type="text" name="name" placeholder="Name" formControlName="name">  
<p *ngIf="form.get('name').hasError('minlength')" class="error">Minimum 3 characters.</p>  
<p *ngIf="form.get('name').hasError('required')" class="error">Required.</p>
```


Model driven form – Custom validator

- Het is ook mogelijk om custom validators te maken:

```
//File: email.validator.ts

import { FormControl } from '@angular/forms';

export function validateEmail(control: FormControl) {
  if (typeof (control.value) === 'string') {
    if (control.value.includes('@')) {
      return null;
    } else {
      return { validateEmail: { valid: false } };
    }
  }
}
```

Model driven form – Custom validator

- Email validator gebruiken in de FormGroup:

```
//File: modelform.component.ts
import { validateEmail } from './email.validator';

this.myForm = new FormGroup({
  first: new FormControl('Dries', []),
  last: new FormControl('Swinnen', []),
  email: new FormControl(null, [ validateEmail ]),
```

Model driven form – SubFormGroups

- Het is mogelijk om een formulier op te splitsen in verschillende SubFormGroups.
- Hierdoor krijgt je `this.myForm.value` aparte objecten voor de subFormGroup:

```
<form [formGroup]="myForm">
First name: <input type="text" FormControlName="first"> <br/>
Last name: <input type="text" FormControlName="last"> <br/>
Email: <input FormControlName="email"> <br/>
<div FormGroupName="address">
Address: <input FormControlName="street"> <br/>
City: <input FormControlName="city"> <br/>
</div>
<input type="button" value="Submit" (click)=onSubmit() [disabled]="!myForm.valid" />
</form>
```

```
Submitted:{
  "first": "Dries",
  "last": "Swinnen",
  "email": "qzefoij@qzefoij",
  "address": {
    "street": "zqefoij",
    "city": "zefo"
  }
}
```

Model driven form – SubFormGroups

- Het is mogelijk om een formulier op te splitsen in verschillende SubFormGroups.
- Hierdoor krijgt je `this.myForm.value` aparte objecten voor de subFormGroup:

```
this.myForm = new FormGroup({  
  first: new FormControl('Dries', []),  
  last: new FormControl('Swinnen', []),  
  email: new FormControl(null, []),  
  address: new FormGroup({  
    street: new FormControl(null, []),  
    city: new FormControl(null, [])  
  })  
});
```

```
Submitted:{  
  "first": "Dries",  
  "last": "Swinnen",  
  "email": "qzefoij@qzefoij",  
  "address": {  
    "street": "zqzefoij",  
    "city": "zefo"  
  }  
}
```

Model driven forms - FormBuilder

- Een FormBuilder kan gebruikt worden om je code te vereenvoudigen
 - Aanspreken van constructor FormControls niet meer nodig.

```
export class FormBuilderComp {  
  form: FormGroup;  
  
  constructor(private fb: FormBuilder) {  
    this.form = fb.group({  
      name: fb.group({  
        first: ['Nancy', Validators.minLength(2)],  
        last: 'Drew',  
      }),  
      email: '',  
    });  
  }  
}
```