Databinding & eventbinding

Angular

HOGESCHOOL PXL

Databinding

- Elementen en events kunnen gekoppeld worden aan:
  - Code in de component klasse
  - Properties (in de component klasse)
- Meerdere syntaxen mogelijk, uitwerking is anders!
  - One way binding
  - Two way binding

Databinding

Type binding	Omschrijving	Syntax
One-way binding Controller to view	Output van een variabele (property) in de component klasse (of andere elementen in de view) tonen in de view	{{ variable_name }}
Two-way binding	Waardes uit een input veld koppelen aan een variabele (property) in de component klasse	[(ngModel)]="variable_name"
One-way binding View to controller	View events koppelen aan een functie uit de component klasse	(click)='voorbeeldmethode()'
One-way binding View to controller	Eigenschappen van een element koppelen aan een variabele (property) van de component klasse (of andere waardes in de view).	[disabled]="isDisabled"

One-way output binding

- Interpolatie
- Output data van de component klasse naar de view
- Double brackets syntax

```
{{ expressie }}
```

One way output binding

- De expressie binnen de accolades kan een aantal zaken bevatten:
  - Variabele

```
{{ contact.name }}
```

  - Expressie

```
{{ a + b }}
```

  - Functie

```
{{ getMessage() }}
```

  - String

```
{{ ‘hallo wereld’ }}
```

One way output binding

- Variabele staan meestal in de component klasse
- Kunnen soms ook verwijzen naar andere elementen in de view

```
1   <form #f="ngForm">
2   <input name="first" ngModel required #first="ngModel"/>
3   </form>
4   <p>First name value: {{ first.value }}</p>
5   <p>Form valid: {{ f.valid }}</p>
```

One way output binding: voorbeelden

```
  template: `<h4>String</h4>
  <p>Er zijn {{days}} {{unit}} in een jaar!</p>
  <h4>Datum</h4>
  <p>{{today|date: 'dd/MM/yy'}}</p>
  <h4>Tekst</h4>
  <p>{{text}}</p>`,
  styleUrls: ['./output.component.css']
})
export class OutputComponent {
  text: string = 'Lorem ipsum dolor sit amet';
  days: number = 365;
  unit: string = 'dagen';
  today: Date = new Date();
}
```

String

Er zijn 365 dagen in een jaar!

Datum

01/12/17

Tekst

Lorem ipsum dolor sit amet

One way output binding: voorbeelden

- Bij de eerste reeks worden de variabelen in een zin geplaatst

```
<p> Er zijn {{days}} {{unit}} in een jaar!</p>
```

- Bij het 2de voorbeeld wordt er een pipe gebruikt om de datum property een format mee te geven:
  - Pipes worden later verder behandeld

```
{{today|date: ‘dd/MM/yy’ }}
```

- Alle variabelen worden gedefinieerd in de outputComponent klasse

```
text: string = ‘Lorem ipsum …’;
days: number = 365;
unit: string = ‘dagen’;
today: Date = new Date(); //Let op de hoofdletter in het datatype
```

Two-way input binding

- In 2 richtingen
- Composite Bracket syntax
  - [(ngModel)] = “variabeleNaam”;
- Wordt gebruikt bij input velden
- Voorbeeld:

```
<input type=“tekst [(ngModel)]=“first_name”>
```

```
export class exampleComponent {
    first_name: string = “Dries Swinnen”;
}c
```

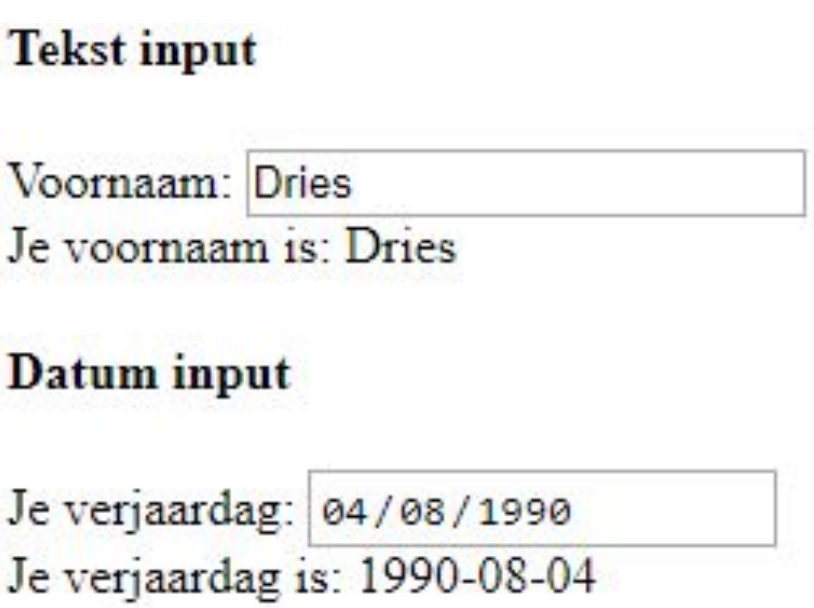
Two-way input binding

- Om two-way binding te gebruiken, moet je de FormsModule toevoegen aan de app.module.ts file!!

```
import { FormsModule } from '@angular/forms';
@NgModule({
declarations: [
    AppComponent,
    OutputComponent,
  ],
imports: [
    BrowserModule,
    FormsModule,
  ],
providers: [],
bootstrap: [AppComponent]})
```

Two-way input binding: voorbeelden

```
  template: `<h4>Tekst input</h4>
  Voornaam: <input type="text"
  [(ngModel)]="voornaam"/><br/>
  Je voornaam is: {{voornaam}}
  <h4>Datum input</h4>
  Je verjaardag: <input type="date"
  [(ngModel)]="verjaardag"/><br>
  Je verjaardag is: {{verjaardag}}`,
  styleUrls: ['./input.component.css']
})
export class InputComponent {
    voornaam: string;
    verjaardag: Date;
}
```



Event binding

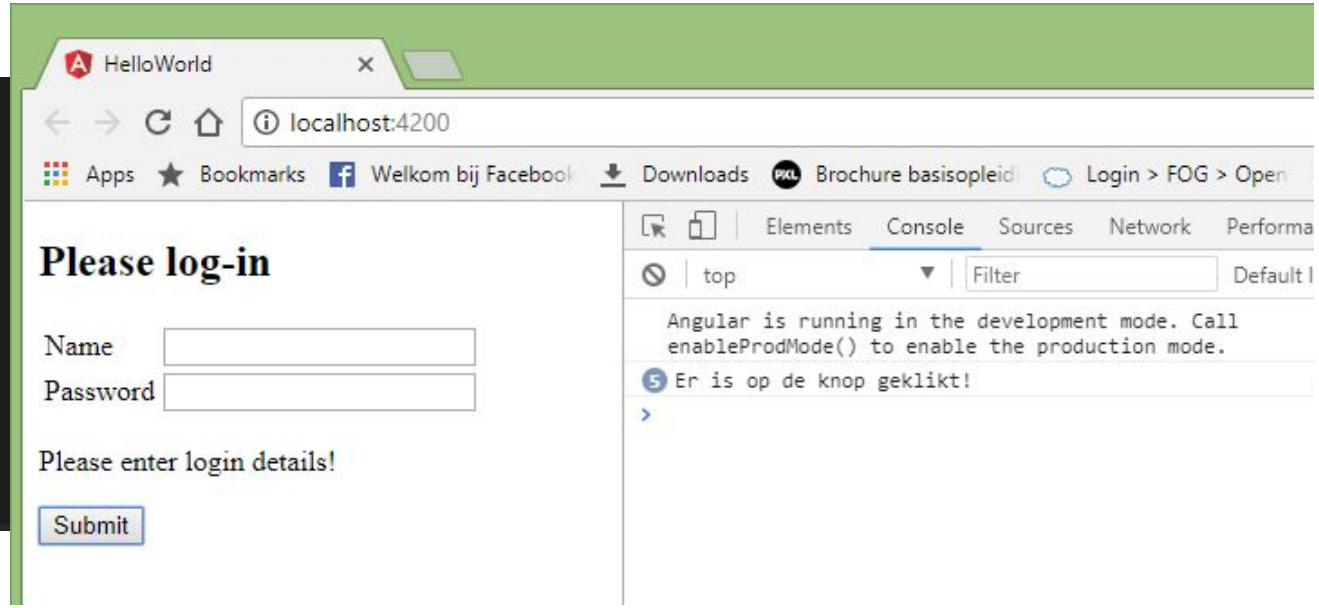
- Functies uit de Component klasse kunnen gekoppeld worden aan DOM events
  - Verschillende events zoals: submit, click, dblclick, dragover, focus, blur, keydown, keyup …
- One way databinding van view naar model

```
13    <button (click)="verwerk()">Submit</button>
```

Event binding

- Koppeling kan op eender welk element
  - Ook op components!

```
8   export class LoginComponent {
9       name: string;
10      password: string;
11      message: string = 'Please enter login details!';
12
13      verwerk(){
14          console.log('Er is op de knop geklikt!');
15      }
16  }
```

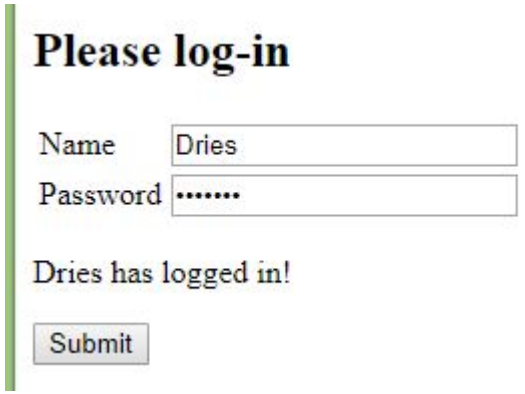


HOGESCHOOL PXL

Event handling & databinding

- In onderstaand voorbeeld wordt message aangepast.
- Name heeft automatisch de waarde van het input veld door [(ngModel)]) **(2 way databinding)**
- Message wordt automatisch aangepast op de view door de verwijzing {{ message }} (**1 way databinding – model to view**)

```
8   export class LoginComponent {
9       name: string;
10      password: string;
11      message: string = 'Please enter login details!';
12
13      verwerk(){
14          this.message = this.name + ' has logged in!';
15      }
16  }
```

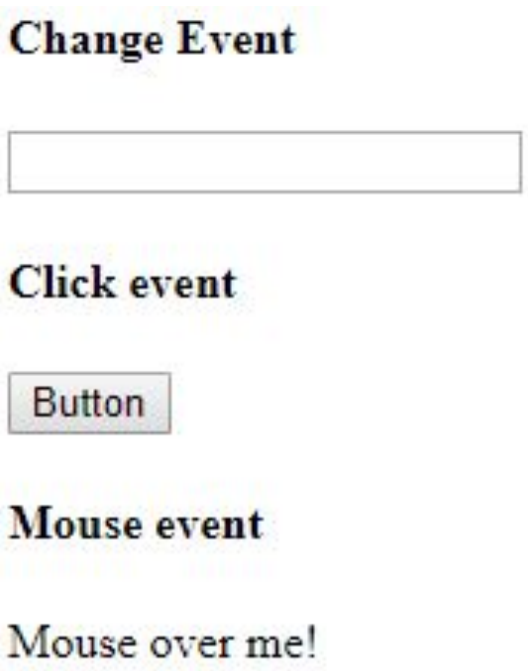


HOGESCHOOL PXL

Event handling: voorbeelden

```
  template:
  <input id="txt1" type="text"
  (keydown)="toggle('txt1')" [(ngModel)]="voornaam">
  <button id="btn1" (click)="toggle('btn1')">
  Button</button>
  <div id="div1" (mouseover)="toggle('div1')"
  (mouseleave)="toggle('div1')">Mouse over me!</div>
  `,
  styleUrls: ['./event.component.css']
})
export class EventComponent {
  toggle(id) {
    let e = document.getElementById(id);
    let bc = e.style.backgroundColor;
    bc = (bc != 'lightblue') ? 'lightblue' : 'yellow';
    e.style.backgroundColor = bc;
  }

}
```



Data doorgeven met @Input()

- Bij geneste components kan er data doorgegeven worden **van parent naar child** component
- Onderstaand voorbeeld: Parent en Child zijn onafhankelijk van elkaar

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
      <h1>Parent</h1>
      <app-child></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

```
// child.component.ts
@Component({
  selector: 'app-child',
  template: `
      <h2>Child component</h2>
      <p>{{msg}}</p>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'child string';
}
```

Data doorgeven met @Input()

- @Input() annotatie wordt toegevoegd aan een property in de component Klasse van de child component
- Zorgt ervoor dat de variabele msg gevuld wordt via property binding vanuit de parent
- Input import voorzien

```
import { Component, OnInit, Input } from '@angular/core';

// child.component.ts
@Component({
  selector: 'app-child',
  template: `
      <h2>Child component</h2>
      <p>{{msg}}</p>
    `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @Input() msg: string;
}
```

Data doorgeven met @Input()

- In de parent component voorzien we in de declaratie van de child component:
  - Koppeling naar de input variabele in de child -- [msg]
  - Koppeling naar de variabele in de parent -- msgText

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
      <h1>Parent</h1>
      <app-child [msg]="msgText"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

Data doorgeven met @Input()

```ts
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
      <h1>Parent</h1>
      <app-child [msg]="msgText"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

```ts
import { Component, OnInit, Input } from '@angular/core';

// child.component.ts
@Component({
  selector: 'app-child',
  template: `
      <h2>Child component</h2>
      <p>{{msg}}</p>
    `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @Input() msg: string;
}
```

Data doorgeven met @Output()

- Bij geneste components kan er data doorgegeven worden **van child naar parent** component
- Onderstaand voorbeeld: Parent en Child zijn onafhankelijk van elkaar

```ts
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
      <h1>Parent</h1>
      <app-child></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

```ts
// child.component.ts
@Component({
  selector: 'app-child',
  template: `
      <h2>Child component</h2>
      <button (click)="onChange()">Click me</button>
    `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'message from the child';

  onChange() {
    console.log('Button clicked');
  }
}
```

Date doorgeven met @Output()

- Binnen de child component wordt er een eventEmitter aangemaakt.
- Deze property krijgt de @Output() annotatie.
- Import voor eventEmitter en @Output() voorzien

```
import { Component, Output, EventEmitter } from '@angular/core';

// child.component.ts
@Component({
  selector: 'app-child',
  template: `
      <h2>Child component</h2>
      <button (click)="onChange()">Click me</button>
    `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'Message from the child';
  @Output() childValueChange = new EventEmitter();

    onChange() {
      this.childValueChange.emit(this.msg);
    }
}
```

Data doorgeven met @Output()

- In de Parent Component wordt de childValueChange event gekoppeld aan een methode in de parent component klasse
- Methode krijgt $event als argument

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
      <h1>Parent</h1>
      <app-child (childValueChange)="eventParent($event)">
      </app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {

  eventParent(event: string){
    console.log('event from child:' + event);
  }
}
```

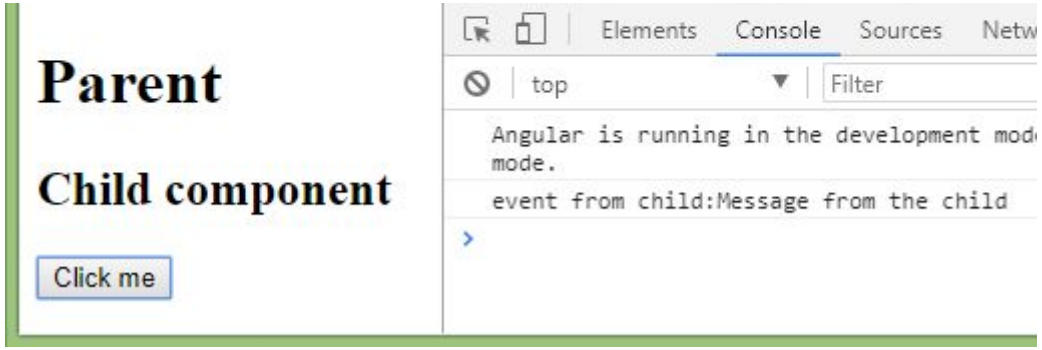
Data doorgeven met @Output()

```ts
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
      <h1>Parent</h1>
      <app-child (childValueChange)="eventParent($event)">
      </app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {

  eventParent(event: string){
    console.log('event from child:' + event);
  }
}
```

```ts
// child.component.ts
@Component({
  selector: 'app-child',
  template: `
      <h2>Child component</h2>
      <button (click)="onChange()">Click me</button>
    `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'Message from the child';
  @Output() childValueChange = new EventEmitter();

    onChange() {
      this.childValueChange.emit(this.msg);
    }
}
```



Data doorgeven met @Output()

- $event
  - parameter wordt altijd meegegeven aan de callback methode
  - Kan eender welk datatype bevatten
    - String, number, Date, Object, Person, Contact, …