

# Web Frameworks Les 2

## Inleiding tot Angular

*Kristof Michiels (team: Sven Mariën, Andie Similon)*

# Topics

- Angular in vogelvlucht + voordelen
- Tools & setup: Node, npm, Angular CLI, VSCode
- Structuur van een Angular-app
- Componenten: opbouw, subcomponenten, herbruikbaarheid
- Data binding: interpolation, property, event, two-way
- Control flow: `@if` , `@for` , `@switch`

# Angular in vogelvlucht

# Angular in vogelvlucht

- **Google's front-end framework** (open-source, enterprise-grade)
- Ontstaan als **AngularJS (2010)** → vernieuwd tot **Angular 2+ (2016)** in TypeScript
- **Angular Renaissance (vanaf v16, 2023):**
  - Standalone components → minder boilerplate
  - Nieuwe template control flow ( `@if` , `@for` )
  - **Signals** → modern reactivity model

# Angular in vogelvlucht (2)

- Vandaag **Angular 20 (2024)**
  - Volledig ecosysteem (CLI, Material, RxJS)
  - Focus op **schaalbaarheid, duurzaamheid en moderne best practices**

# Waarom Angular?

- **Sterke structuur** → opinionated, duidelijke richtlijnen
- **Component-based architecture** → herbruikbare bouwstenen
- **Volledige TypeScript-integratie** → type safety & productiviteit
- **CLI & tooling** → snelle setup, scaffolding, testing
- **Brede community + enterprise support** → duurzaam & betrouwbaar

# Waarom ik Angular kies

- **Respecteert webstandaarden** → toekomstbestendig
- **Semantische HTML** → toegankelijkheid & SEO ingebakken
- **Vrijheid in styling** → native CSS, SCSS, Tailwind, ...
- **Ontworpen voor het web** → breed compatibel
- **Renaissance (v17+)** → standalone components, minder boilerplate, vlottere dev experience

👉 Mijn overtuiging: **Angular maakt je een betere developer**

# Tools & setup



# Noodzakelijke tools

- **Node.js** → runtime (laatste stabiele versie)
- **npm** → package manager
- **Angular CLI** → via npm geïnstalleerd
- **Visual Studio Code** → editor
- **Angular Language Service plugin** → betere code hints in VSCode

# De omgeving opzetten

- Werken dus in een **Node-context** (bekend van Web Programming)
- Check installatie:

```
node -v  
npm -v
```

- Installatie van Angular CLI

```
npm install -g @angular/cli  
ng version
```

# De Angular CLI

- Command Line Interface voor Angular-projecten
- Zorgt voor:
  - **Scaffolding** (structuur en boilerplate code)
  - **Development server** met live reloading
  - **Generators** voor components, services, modules
  - **Build & deploy tools**

# Belangrijkste CLI-commando's

- `ng new <project>` → nieuw Angular-project
- `ng serve` → start dev server (met live reload)
- `ng generate component <naam>` → maakt component
- `ng generate service <naam>` → maakt service
- `ng build` → bouwt productieklare versie

# Waarom de CLI gebruiken? 🚀

- **Snelle setup** → geen gedoe met configuratie
- **Consistente structuur** → iedereen start gelijk
- **Productiviteit** → scaffolding, testen & deploy ingebouwd
- **Enterprise-ready** → volgt Angular best practices automatisch

# Structuur van een Angular applicatie

# Structuur van Angular-apps



- **package.json** → dependencies & scripts
- **angular.json** → Angular CLI configuratie
- **tsconfig.json** → TypeScript instellingen
- **src/** → hier leeft je app-code

# Belangrijke bestanden

- **main.ts** → entry point, bootstrapt de app
- **index.html** → hostpagina
- **styles.css** → globale stijlen
- **public/** → statische assets



# De app-folder

- **app/** → kern van je toepassing
    - `app.component.*` → root component
    - `app.routes.ts` → navigatie / router configuratie
    - Verdere componenten & modules bouwen hierop voort
- 👉 Alles start bij de **app-component** (root)

# Componenten: de bouwstenen van Angular

# Wat is een component?

- Een **component** = de kleinste bouwsteen van een Angular-app
- Bestaat uit 3 delen:
  - **TypeScript** → logica & state
  - **Template (HTML)** → weergave
  - **Styles (CSS)** → opmaak, scoped
- Elke app start bij de **root component**  
( AppComponent )

# Wat is een component?

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

@Component({
  selector: 'app-hallo',
  template: '<h1>Hallo {{ naam }}</h1>',
  styles: ['h1 { color: darkblue; }']
})
export class HalloComponent {
  naam = 'Kristof Michiels';
}
```

# Templates & Styling

- **Inline template** (zoals hierboven) of **apart bestand**

```
@Component({  
  templateUrl: './hallo.component.html',  
  styleUrls: ['./hallo.component.css']  
})
```

- Scoped styling: CSS geldt enkel voor die component
- Volledige vrijheid: CSS, Tailwind, Angular Material
- Templates = UI declaratief beschrijven

# Componenten genereren ⚡

```
ng generate component hallo  
# of korter  
ng g c hallo
```

- Angular CLI maakt componenten automatisch aan en garandeert structuur
- CLI creëert:
  - `hallo.component.ts` → logica
  - `hallo.component.html` → template
  - `hallo.component.css` → styling
  - `hallo.component.spec.ts` → testbestand

# Waar plaats je componenten?

- Standaard in `src/app/`
- Voor grotere apps: organiseer per **feature-map**
  - `app/dashboard/dashboard.component.*`
  - `app/gebruiker/profiel.component.*`
- Hou componenten **klein & overzichtelijk**
  - 1 component = 1 duidelijke verantwoordelijkheid
- Goede structuur = **onderhoudbare applicatie**

# Subcomponenten

- Componenten kunnen **andere componenten bevatten**
- Root ( AppComponent ) kan children renderen via selectors
- Dit maakt Angular-apps **hiërarchisch en modulair**

```
<!-- app.component.html -->  
<h1>Mijn App</h1>  
<app-header></app-header>  
<app-footer></app-footer>
```



# Subcomponenten declareren



- Elke subcomponent moet via import gekend zijn bij ouder
- Werkt op dezelfde manier voor elke child/parent-relatie

```
import { Component } from '@angular/core';
import { HeaderComponent } from './header.component';
import { FooterComponent } from './footer.component';


@Component({
  selector: 'app-root',
  imports: [HeaderComponent, FooterComponent],
  templateUrl: './app.component.html'
})
export class AppComponent {}
```

# Herbruikbaarheid

- Componenten zijn **herbruikbaar** binnen de app
- Best practices:
  - Houd componenten **klein & herbruikbaar**
  - Gebruik **inputs/outputs** om data door te geven
  - Plaats gedeelde componenten in `shared/` folder

# Data binding

# Data binding

- Verbindt **component (TS)**  **template (HTML)**
- Vier vormen van binding:
  1. **Interpolation** → variabele tonen
  2. **Property binding** → attribuut koppelen
  3. **Event binding** → reageren op events
  4. **Two-way binding** → lezen én schrijven tegelijk

# Interpolation (1) ✨

- Plaats de waarde van een variabele rechtstreeks in de HTML
- Notatie: `{{ expressie }}`

## Template

```
<p>Hallo {{ gebruiker }}</p>
```

## Component (TS)

```
export class AppComponent {  
  gebruiker = 'Kristof';  
}
```

# Interpolation (2)

- Kan ook expressies bevatten (geen statements!)

## Template

```
<p>2 + 2 = {{ 2 + 2 }}</p>
```

```
<p>Gebruiker karakters = {{ gebruiker.length }}</p>
```

## Component (TS)

```
export class AppComponent {  
  gebruiker = 'Kristof';  
}
```

# Interpolation (3)

- Kan gebruikt worden in attributen (maar let op: dit is eigenlijk property binding)

## Template

```
<h1 title="{{ gebruiker }}">Welkom!</h1>
```

## Component (TS)

```
export class AppComponent {  
  gebruiker = 'Kristof';  
}
```

# Property binding (1)

- Verbindt een **property van een DOM-element** aan een variabele
- Notatie: `[property]="expression"`

## Template

```
<img [src]="afbeeldingUrl">
```

## Component (TS)

```
export class AppComponent {  
  afbeeldingUrl = 'assets/angular-logo.png';  
}
```



# Property binding (2)

- Ook bruikbaar op custom componenten

## Template

```
<app-gebruiker-card [gebruiker]="actieveGebruiker"></app-gebruiker-card>
```

## Component (TS)

```
export class AppComponent {  
  actieveGebruiker = { id: 1, naam: 'Emily' };  
}
```

# Property binding (3)

- We noemen dit component interaction
- @Input() → dataflow van ouder → kind (parent → child)

```
<!-- gebruiker-card.component.html -->
<div class="card">
  <h3>Gebruiker: {{ gebruiker?.naam }}</h3>
  <p>ID: {{ gebruiker?.id }}</p>
</div>
```

# Property binding (4)

```
// gebruiker-card.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-gebruiker-card',
  templateUrl: './gebruiker-card.component.html',
  styleUrls: ['./gebruiker-card.component.css']
})
export class GebruikerCardComponent {
  @Input() gebruiker: { id: number; naam: string } | null = null;
}
```

# Property binding (5)

- Let op verschil met interpolation
- Property binding is krachtiger en veiliger

## Template

```
<!-- Interpolation -->  
  
  
<!-- Property binding -->  
<img [src]="afbeeldingUrl">
```

## Component (TS)

```
export class AppComponent {  
  afbeeldingUrl = 'assets/angular-logo.png';  
}
```

# Event binding (1) ⚡

- Laat Angular reageren op events
- Notatie: `(event)="handler()"`

## Template

```
<button (click)="zegHallo()">Klik op mij!</button>
```

## Component (TS)

```
export class AppComponent {  
  zegHallo() {  
    alert('Hou je ondertussen al van Angular!');  
  }  
}
```

# Event binding (2)

- Alle DOM-events worden ondersteund

## Template

```
<input (input)="onInput($event)">
<p>Je typte: {{ huidigeWaarde }}</p>
```

## Component (TS)

```
export class AppComponent {
  huidigeWaarde = '';

  onInput(event: Event) {
    const target = event.target as HTMLInputElement;
    this.huidigeWaarde = target.value;
  }
}
```

# Event binding (3)

- Event binding op custom componenten (via @Output)
- Terug die component interaction (andere richting)

## Child template

```
<button (click)="verwittig.emit('Er is op de knop geklikt!')">Parent laten weten</button>
```

## Child TS

```
@Output() verwittig = new EventEmitter<string>();
```

# Event binding (4)

- @Output() → eventflow van kind → ouder (child → parent)

## Parent template

```
<app-child (verwittig)="onVerwittig($event)"></app-child>
```

## Parent TS

```
onVerwittig(message: string) {  
    console.log('Child zegt:', message);  
}
```



# Two-way binding (1)

- Combineert property + event binding
- Notatie: `[(ngModel)]="property"`
- Handig bij formulieren: waarde in de input = altijd gesynchroniseerd met de component
- We importeren hiervoor de forms-module in het ts-bestand

## Template

```
Jouw gebruikersnaam: <input [(ngModel)]="gebruiker">
<p>Hallo {{ gebruiker }}</p>
```

# Two-way binding (2)

## Component (TS)

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

@Component({
  selector: 'app-gebruiker',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './gebruiker.component.html',
  styleUrls: ['./gebruiker.component.css']
})
export class GebruikerComponent {
  gebruiker = "Angèle";
}
```

# Control flow

# Control flow in templates

- Sinds Angular 16: nieuwe **template control flow**
- Maakt de logica in je HTML **duidelijker en leesbaarder**

## Voorbeeld:

```
@if (isIngelogd) {  
  <p>Welkom terug!</p>  
} @else {  
  <p>Inloggen alstublieft</p>  
}
```

# Control flow in templates (2)

## Component (TS)

```
export class AppComponent {  
  isIngeologd = false;  
}
```

# Lijsten tonen met @for

- Herhaal een element voor elk item in een array
- `track` helpt Angular efficiënt updaten

## Template

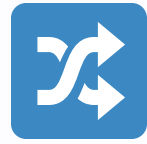
```
<ul>
  @for (item of items; track item.id) {
    <li>{{ item.naam }}</li>
  }
</ul>
```

# Lijsten tonen met @for (2)

## Component (TS)

```
export class AppComponent {  
  items = [  
    { id: 1, naam: 'Laptop' },  
    { id: 2, naam: 'Smartphone' },  
    { id: 3, naam: 'Tablet' }  
  ];  
}
```

# Keuzes maken met @switch



- Handig bij meerdere opties

## Template

```
@switch (rol) {  
  @case ('admin') { <p>Admin panel</p> }  
  @case ('user') { <p>User dashboard</p> }  
  @default { <p>Guest view</p> }  
}
```

## Component (TS)

```
export class AppComponent {  
  rol = 'user';  
}
```



# Waarom control flow gebruiken?



- **Declaratief en overzichtelijk** in de template
- Houdt de logica **dicht bij de UI**
- Vervangt de oudere `*ngIf` , `*ngFor` , `*ngSwitch` syntaxis
- Zorgt voor minder fouten en duidelijkere code

# Jouw taak voor deze week

- Labo's beginnen in week 2
- Je gaat je deze week vertrouwd maken met de Angular CLI
- Je gaat wat componenten aanmaken
- Je gaat experimenteren met data binding en control flow
- Op deze manier kunnen we volgende week goed beginnen met oefenen

**Bedankt voor je aandacht! 😊**

**Vragen? Feedback?**

**kristof.michiels01@ap.be**