Angular

Was ist Angular?

Angular ist ein Open-Source-Framework, das zur Entwicklung von Webanwendungen und Single-Page-Applications (SPA) verwendet wird.

Wesentliche Merkmale sind:

1. **Komponenten basiert** → Anwendungen werden in wiederverwendbare Komponenten unterteilt

2. **Nutzt Typescript** → Angular nutzt die typisierte JavaScript-Variante für bessere Wartbarkeit

3. **Entwicklertools** → Angular verfügt über eine Kommandozeilenoberfläche für schnelles und einfaches erstellen/testen der Anwendung(Angular CLI)

4. **Two-way-data binding** → Synchronisiert daten automatisch zwischen logik und View

5. **Dependency Injection** → Vereinfacht Management von Abhängigkeiten zwischen verschiedenen Komponenten

Jede Komponente besteht aus:

- Typescript(Logik)
- HTML(Template)
- CSS(Design)
- Spec.ts(Unit test)

Generelles Setup

- Installieren von node/npm/nvm
- Installieren der Angular CLI
- Neues Projekt anlegen und loslegen (Gewünschte IDE sollte ebenfalls vorhanden sein)

Folgende Schritte werden hierfür benötigt (empfohlen):

1. Nvm(node version manager) installieren: '***curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.2/install.sh | bash***'
2. Node installieren via NVM: '***nvm install --lts***'
3. Überprüfen der Version: '***node -v***'
4. Installieren von der Angular CLI: '***npm install -g @angular/cli***'
5. Überprüfen der Version: '***ng version***'
6. Neues Projekt anlegen: '***ng new filename***'
7. Projekt starten: '***cd filepath/filename'*** **und starte mit '*ng serve –o*'** (die flag **–open** öffnet das Projekt direkt im Browser)
8. Öffne dein Projekt mit deiner IDE (falls VSCode genutzt wird, einfach zum richtigen Ordner navigieren und im Terminal '**code .**' eingeben, VSCode wird automatisch geöffnet)

Grundlegende Funktionen/Direktiven

**ngFor*

Wird im Template verwendet und funktioniert wie eine ***for*-schleife.**
Beispiel:

```
HTML
<div *ngFor="let item of items"></div>
```

**ngIf*

Wird im Template verwendet und funktioniert wie ein ***if*-statement.**
Beispiel:

```
HTML
<div *ngIf=”true”></div>
```

Template Binding

Wird im Template verwendet und lässt **Werte/Variablen** im Template erscheinen.

Beispiel:

```HTML
<div>{{ variable }}</div>
```

Two-Way data binding

Wird im Template verwendet und aktualisiert Daten zwischen Logik und Template.
Beispiel:

```HTML
<input type="text" [(ngModel)]=”someLogik”>
```

@Input()

Ermöglicht Datenübertragung zu den ‘kind’ Komponenten und wird in Template verwendet, aber im ts deklariert.
Beispiel:

```HTML
<app-child [inputName]=”overrideDefaultName”></app-child>
<!--TS datei der Kind Komponente-->
@Input() inputName: string = ‘Standard Name’;
<!--TS datei Komponente-->
overrideDefaultName: string = ‘Anderer Name’
```

Erstellen/Benutzung einer Komponente

Zum Erstellen einer neuen Komponente wird die Angular CLI verwendet.
Dazu kann folgender Befehl ausgeführt werden:

ng generate component nameDerKomponente
oder in der Kurzform:
ng g c nameDerKomponente

Angular legt dabei automatisch einen neuen Ordner für die Komponente an und erstellt die zugehörigen Dateien für **Template (HTML)**, **Styles (CSS/SCSS)**, **Logik (TypeScript)** sowie **Unit Tests (Spec)**.
Zusätzlich wird die Komponente automatisch im entsprechenden Modul registriert, in der Regel im app.module.ts oder in einem Feature-Modul.

Beispiel Projektstruktur:

```
None
my-angular-app/
├── angular.json
├── package.json
├── tsconfig.json
├── src/
│   ├── index.html
│   ├── main.ts
│   ├── styles.scss
│   └── app/
│      ├── app.component.ts
│      ├── app.component.html
│      ├── app.component.scss
│      ├── app.module.ts
│      │
│      └── components/
│         ├── header/
│         │   ├── header.component.ts
│         │   ├── header.component.html
│         │   ├── header.component.spec.ts
│         │   └── header.component.scss
│         │
│         └── product-list/
│            ├── product-list.component.ts
│            ├── product-list.component.html
│            ├── product-list.component.spec.ts
│            └── product-list.component.scss
```

In der Typescript-Datei wird die Komponente deklariert:

```TypeScript
@Component({
  selector: 'app-nameDerKomponente',
  templateUrl: './nameDerKomponente.component.html',
  styleUrls: ['./nameDerKomponente.component.css']
})
```

selector definiert, wie die Komponente im HTML verwendet wird.
templateUrl und **styleUrls** verweisen auf externe Dateien.

Der Selektor wird wie ein eigenes HTML-Element verwendet:

```HTML
<app-nameDerKomponente></app-nameDerKomponente>
```

Komponenten Stylen

In der CSS-Datei kann man den verschiedenen Elementen innerhalb dieser Komponente Styles geben.
Um die Elemente auszuwählen, gibt man diesen zunächst eine '*Klasse'* oder eine *'ID'.* Man kann auch das Element selbst auswählen.
Bsp.:

```HTML
<div class="container"></div>
<div id="einzigartigeId"></div>
<!--Element ohne Klasse oder id -->
<div></div>
```

```css
// Übernimmt für alle elemente mit der klasse folgende styles
.container {
  width: 500px;
  height: 100px;
  background-color: #000000;
}

// Übernimmt für alle Elemente mit der ID folgende styles
#einzigartigeId {
  width: 500px;
  height: 100px;
  background-color: #555555;
}

// Übernimmt für alle div elemente folgende styles
div {
  width: 500px;
  height: 100px;
  background-color: #aaaaaa;
}
```

Diese Styles haben außerhalb der Komponente keine Wirkung! Selbst wenn eine andere Komponente die gleiche Klasse oder ID hat!

Ausgabe des Beispiels im Browser:

[ngClass]

Dieses Attribute wird im Template verwendet. Hiermit kann man anhand einer Bedingung klassen auf HTML elemente machen oder entfernen
Beispiel:

```HTML
<div [ngClass]="{ active: isActive, inactive: !isActive }">
  Status: {{ isActive ? 'Aktiv' : 'Inaktiv' }}
</div>
```

[ngStyle]

Dieses Attribute wird im Template verwendet. Hiermit kann man anhand einer Bedingung verschiedene Styles setzen.
Beispiel:

```HTML
<div [ngStyle]="{ color: isActive ? 'green' : 'red' }">
  Status: {{ isActive ? 'Aktiv' : 'Inaktiv' }}
</div>
```