Ionic & Angular

EXXETA
CONSULTING AND TECHNOLOGIES

Native Apps mit Angular entwickeln
Alle Plattformen – eine Codebasis

Jan Buchholz – EXXETA GmbH

Leipzig, 29.03.2019

JUG saxony

Inhalt

1. **Motivation**
2. **Grundlagen Angular**
3. **Grundlagen Ionic**
4. **Vergleich**
5. **Zusammenfassung**
6. **Diskussion**

Motivation

Ein Framework – viele Möglichkeiten

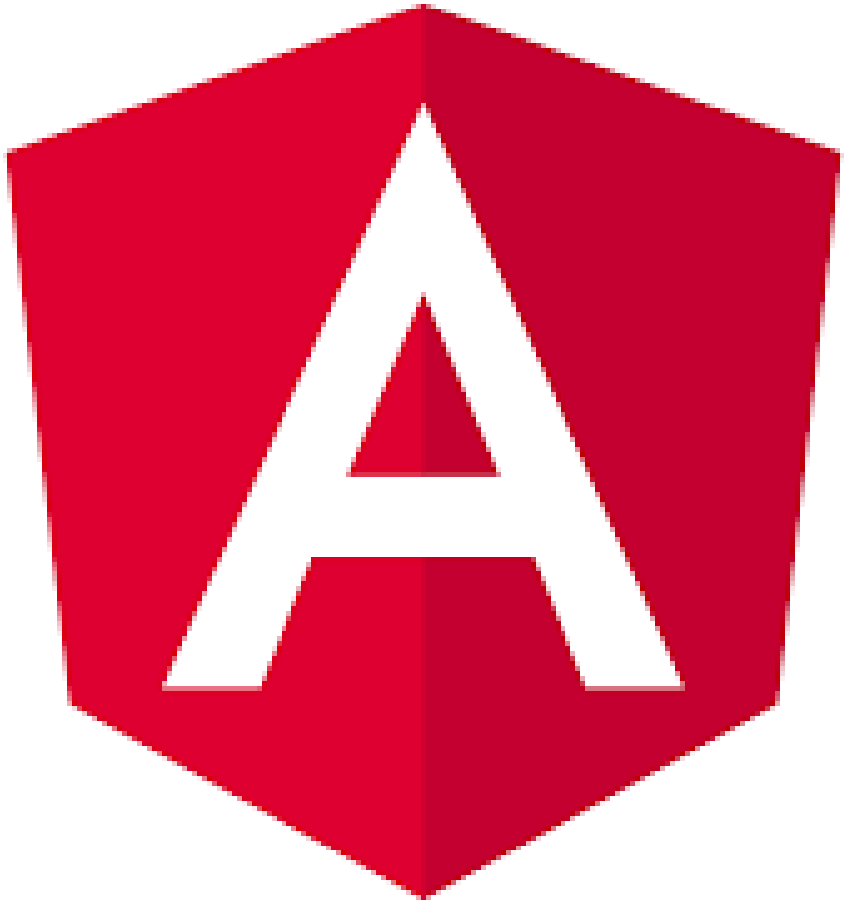
- verschiedene Plattformen setzen verschiedene Programmiersprachen voraus
- Ziel: eine Sprache für alle Plattformen, eine gemeinsame Codebasis
- Wunsch: bekanntes Wissen anwenden

- Ergebnis: wir nutzen Angular (oder alternativ React, Vue, …), um native Apps zu erstellen
  - Ionic übernimmt für uns plattformspezifische Aufgaben
  - Capacitor ist die Brücke zwischen unserer App und der darunterliegenden Plattform

Angular

Grundlagen

- CLI
- Bootstrapping
- Components
- Data binding
- Services / Dependency Injection
- Routing



Angular

CLI & Bootstrapping

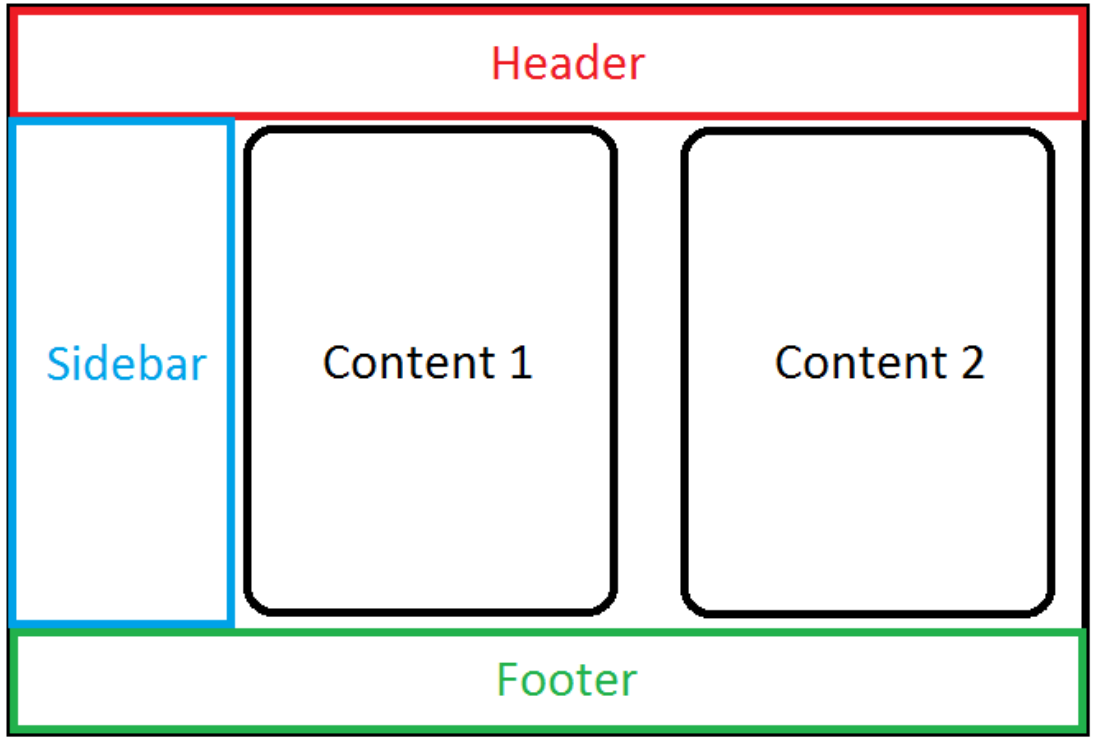
- CLI erleichtert uns die Arbeit mit Angular
  - Konfiguration der Anwendung
  - Anlegen neuer Elemente in der Anwendung
  - Dev-Server / Test / Build
- Bootstrapping
  - index.html
  - main.ts
  - AppModule
  - AppComponent

Angular

Components

- App aus vielen *Components* zusammengesetzt, die zur Laufzeit generiert / ausgetauscht / zerstört werden können

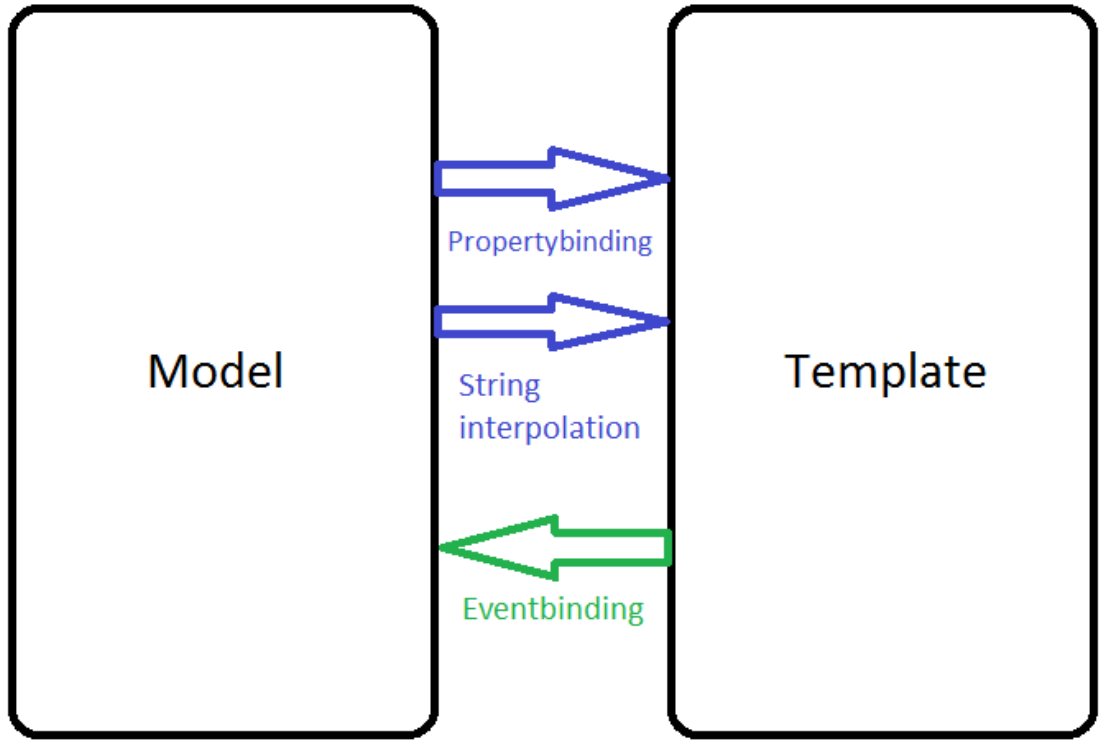
- Components bestehen aus:
  - Template (html)
  - Model (ts)
  - (Styling (css))



Angular

Data Binding

- Template und Model müssen miteinander kommunizieren
  - Template stellt Daten des Models dar
    - Property binding
    - String interpolation
  - Model reagiert auf Events im Template
    - Event binding



Angular

Property Binding

- Template und Model müssen miteinander kommunizieren
  - Template stellt Daten des Models dar
    - Property binding
    - String interpolation

```
this.pokemon = {
  pokedex: 1,
  name: 'Bisasam',
  types: ['Pflanze', 'Gift'],
  pictureUrl: '...'
};
```

```
<img [src]="pokemon.pictureUrl" width="400px">
<h2>Nr. {{pokemon.pokedex}}</h2>
<h1>{{pokemon.name}}</h1>
```

```
<h2 [innerHTML]="'Nr. ' + pokemon.pokedex"></h2>
```

Nr. 1

Bisasam

```
<img _ngcontent-c1 width="400px" src="https://www.pokewiki.de/images/9/96/Sugimori_001.png">
<h2 _ngcontent-c1>Nr. 1</h2>
<h1 _ngcontent-c1>Bisasam</h1>
```

Angular

Event Binding

- Template und Model müssen miteinander kommunizieren
  - Model reagiert auf Events im Template

```
<button type="button"
        [disabled]="!f.form.valid"
        (click)="onSubmit()">Add pokemon</button>
```

```
onSubmit() {
  this.pokemonService.addPokemon(this.pokemon);
  this.router.navigate( commands: ['/pokemon']);
}
```

Angular

Services & Dependency Injection

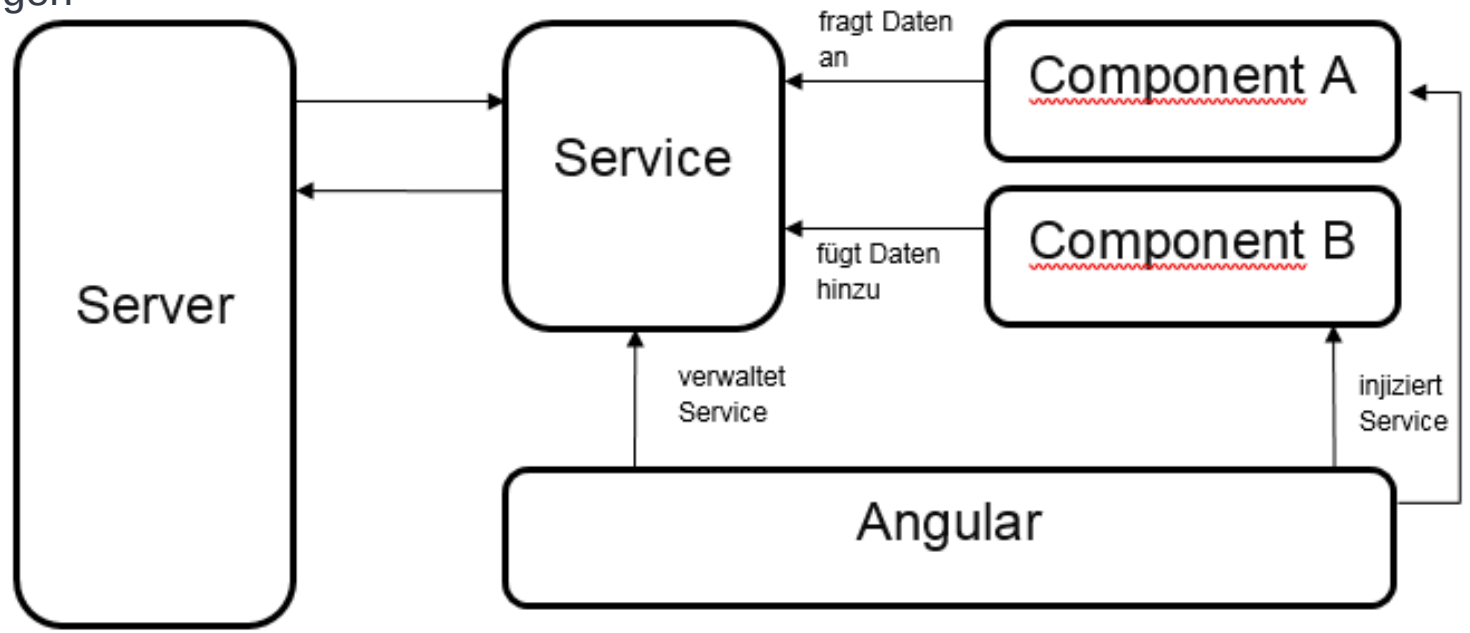
- Services stellen Component-übergreifende Funktionalitäten zur Verfügung
  - Datenbereitstellung / -manipulation
  - Logging …
  - Daten (Pokemon) werden auf einem Server persistiert
  - Service kommuniziert mit dem Server
  - Component A möchte Daten haben, fragt diese beim Service an
  - Component B möchte Daten schreiben, schickt diese zum Service

Angular

Services & Dependency Injection

- Zugriff auf Services erfolgt über DI
  - Angular verwaltet alle Ressourcen für uns
  - mittels C*onstructor Injection* geben wir in der Component an, welche Services wir benötigen

```
constructor(
  private pokemonService: PokemonService,
  private route: ActivatedRoute
) { }
```



Angular

Routing

- Angular stellt *Router* zur Verfügung
  - wir definieren festgelegte Routen (ggf. mit Route-Parametern)
  - wird eine Route aufgerufen, wird die jeweilige Component in die Seite eingebunden
    - durch *Router Outlet* spezifizieren wir die Stelle, an der wir die Component einbauen möchten
  - Aufruf erfolgt mittels *routerLink*

```
{
  path: 'pokemon/:pokedex',
  component: PokemonDetailComponent
}
```

```
<router-outlet></router-outlet>
```

```
▼<jug-root _nghost-c0 ng-version="7.2.8">
    <router-outlet _ngcontent-c0></router-outlet>
  ▶<jug-pokemon-detail _nghost-c1>…</jug-pokemon-detail>
  </jug-root>
```

```
<button type="button"
        routerLink="/pokemon">
  Back to the list
</button>
```

Ionic

Grundlagen

- eine Codebasis für alle Plattformen

- Sammlung von UI-Components, die sich den jeweiligen Plattformen anpassen
- Zugriff auf native Features (Kamera, GPS, …)

- geschichtlich an Angular gebunden, aber inzwischen losgelöst

Ionic

UI Components

- Ionic stellt ≈ 100 Components zur Verfügung
  - ion-button, ion-list, ion-img, …
- außerdem große Sammlung von Icons (≈ 400)

```
<button (click)="onSubmit()">
  <i class="fa fa-add"></i>
</button>

<ion-button (click)="onSubmit()">
  <ion-icon name="add" slot="icon-only"></ion-icon>
</ion-button>
```

- slots definieren bestimmte Areale innerhalb einer Component (start, end, …)
- Erscheinungsbild ist plattformabhängig

Android

iOS

- Styling über globale Variablen und Attribute

Vergleich Angular – Ionic

Gemeinsamkeiten und Unterschiede

- Modelle & Services sind i.d.R. sehr ähnlich
- Verwendung des Angular Routers
- Kommunikation zwischen Model und View

- Pages vs. Components
  - Pages sind auch Components, decken aber immer den gesamten Bildschirm ab
  - werden i.d.R. als eigene Submodule definiert, um Ressourcen zu sparen
- Styling
  - Angular: Design muss selber erstellt werden
  - Ionic: feste Vorgaben im Design, das sich an der jeweiligen Plattform orientiert
  - i.d.R. kompletter Rewrite der Templates notwendig

Routing mittels NavController

teilweise für „schönes“ Routing notwendig, sollte aber sonst vermieden werden

Zusammenfassung

Ionic ist

- eine Sammlung von UI-Komponenten, mit denen sich schnell und einfach „schöne“ Apps erstellen lassen
- (zusammen mit Capacitor) die Brücke zwischen Web-Technologien und Native Apps
- auch ohne Angular nutzbar (Vanilla JS, Vue, …)
- keine Erweiterung von Angular, sondern ein Framework, das sehr gut mit Angular interagiert
- kein Framework zur *nachträglichen* Migration einer Angular-Anwendung in eine native App

Links

https://angular.io

https://ionicframework.com/docs

https://gitlab.com/leonard590/ionic-jug-saxony

WIR VERBINDEN WELTEN