Angular in der Praxis: Do‘s and Don'ts

WJAX 2018

Karsten Sitterberg

Trainer
Autor
Entwickler

https://sitterberg.com/
karsten@sitterberg.com
@kakulty

Typisch(e) Kunden

Enterprise Anwendungen

- Komplexe Anwendungen
  - Erstellung im Bereich > xx PJ
- Lange Lebensdauer erfordert gute Wartbarkeit
  - Firmen sind bereit zu investieren
- Entwickler
  - Viele Entwickler(teams)
  - Unterschiedliches Know-How
  - Produktivität steht im Vordergrund
  - Fachentwickler vs. Infrastrukturentwickler

Angular richtige Wahl für Enterprise

- Opinionated + Full-Stack
  - Liefert vorgegebene Lösungen:
    Routing, Formulare, HTTP, DI
- Ökosystem
  - CLI, NgRx, Angular Material, PWA, Universal, ...
- TypeScript für langlebige Anwendungen
  - Statische Analysierbarkeit
  - Verbessert Refactoring-Möglichkeiten
  - Produktivität: IDE Support (Linter, Code Qualität)

Deutsche Bank

Osram

Citibank

Audi

Lufthansa Systems

Otto

PwC

Avarto

Deutsche Post

Microsoft

Bosch

Terradata

Provinzial

Barnes & Noble

RWE

LVM

Angular richtige Wahl für Enterprise

Firebase

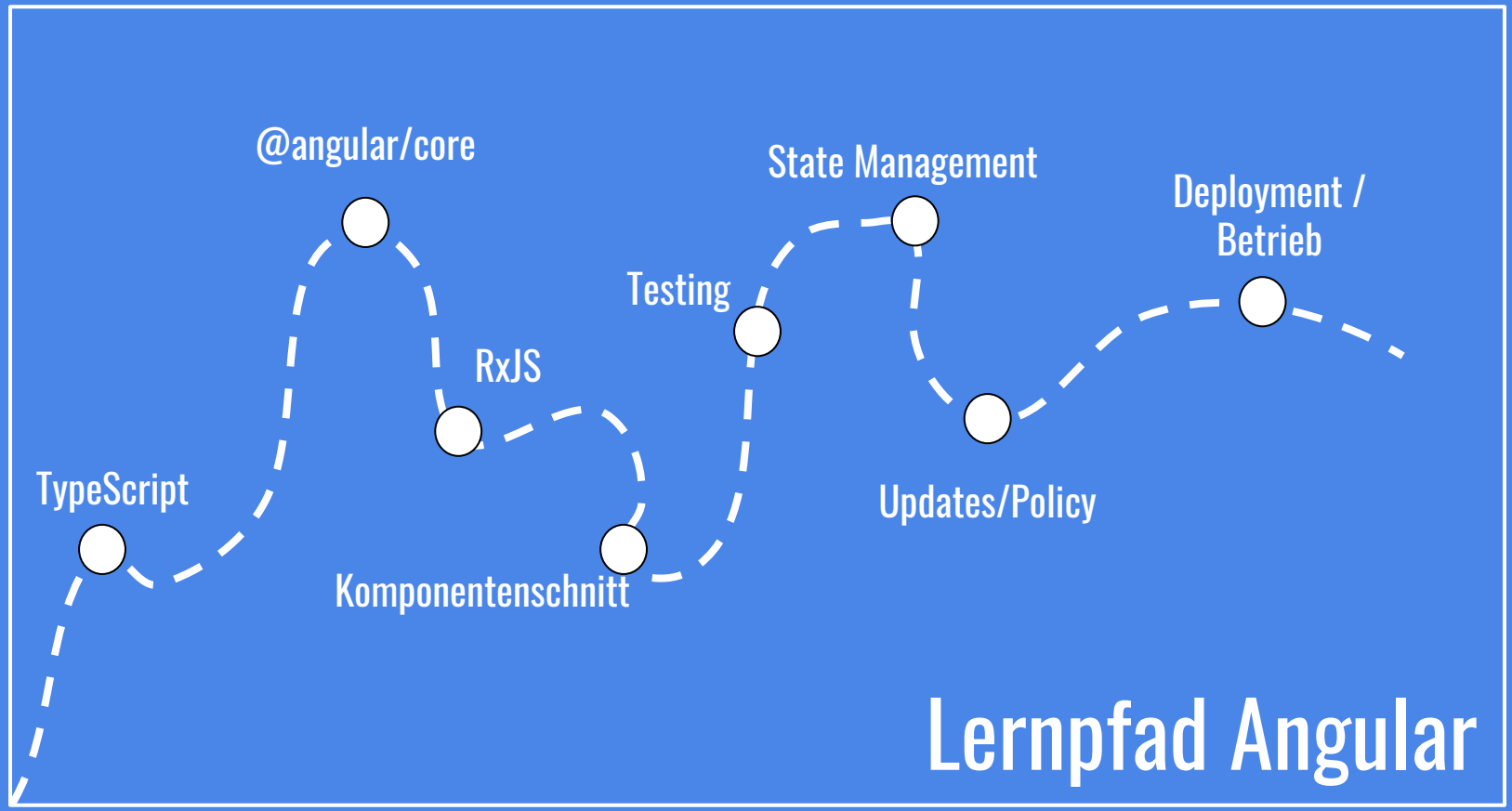
Google Analytics

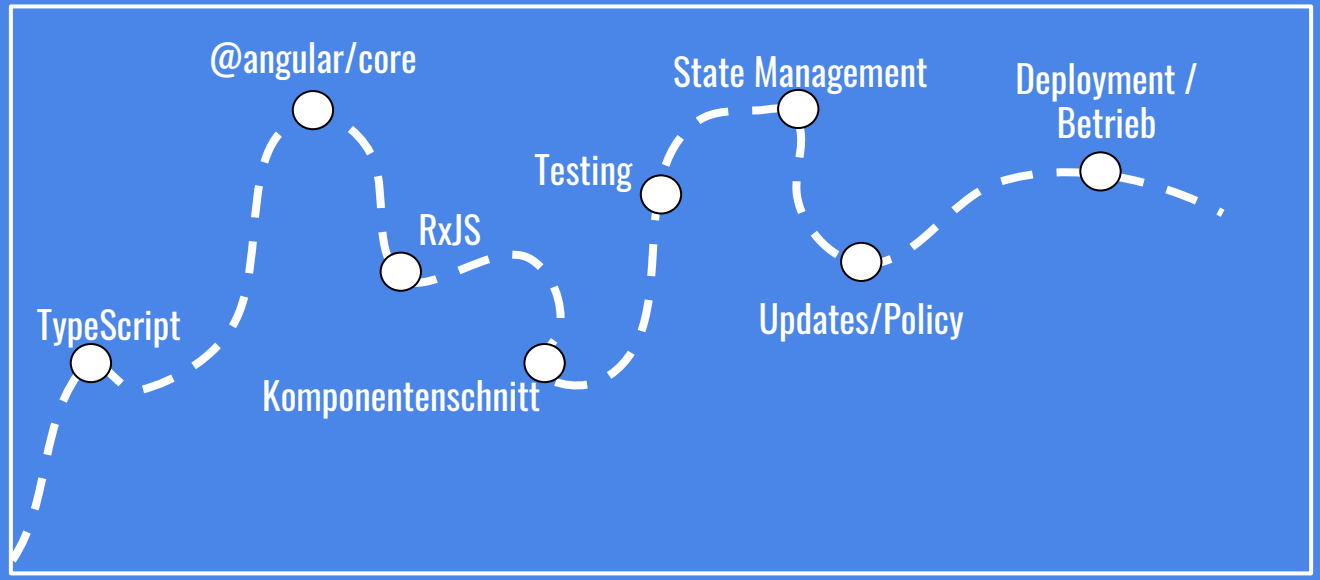
Google Express

Google Cloud Platform

...

+600 more


@angular/core
State Management
Deployment /
Betrieb
Testing
RxJS
TypeScript
Updates/Policy
Komponentenschnitt
Lernpfad Angular



Customizing / Unternehmensvorgaben

Systemschnitt

Elements

API Design

Unternehmens Komponenten

Architektur

Lessons Learned

Beim Standard bleiben

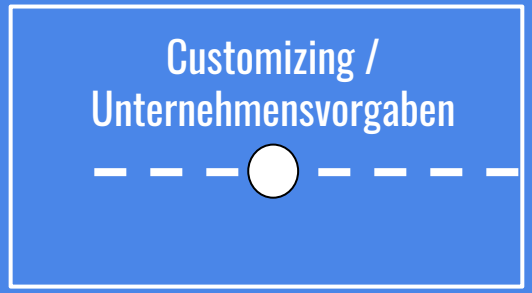
Mit dem Standard beginnen

- Verwendung von Standard Tools
  - Angular CLI
  - Schematics
  - tslint/codelyzer
- Standardeinstellungen
  - Erst Verstehen
  - Dann Customizing

vs.

- Auf gewohnten Mustern bestehen
- Angular verbiegen

Am Standard wachsen



- Best Practises beibehalten
- Firmeninterne Konventionen
  - Eigene Linter-Regeln
  - Eigene Schematics
  - Evtl. eigene Komponenten-Bibliotheken

- Nutzen
  - Interne Unterstützung (Inter-Team-Fähigkeit, Code Reviews)
  - Externe Unterstützung (Code Reviews, Dienstleistung)

TS app.module.ts

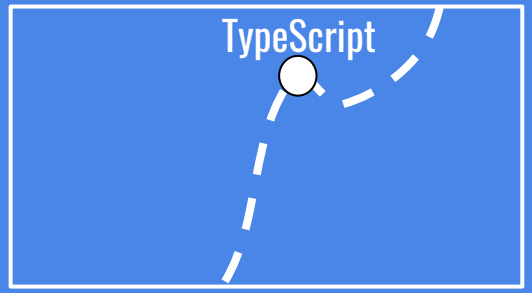
```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Embrace TypeScript



TypeScript

- “JavaScript that scales”
- Statische Typisierung - Statische Codeanalyse
  - Durch Compiler **und** Linter
- Sollte möglichst voll ausgeschöpft werden

Rückgabetypen immer angeben (`void`)

~~`var`, `any`~~

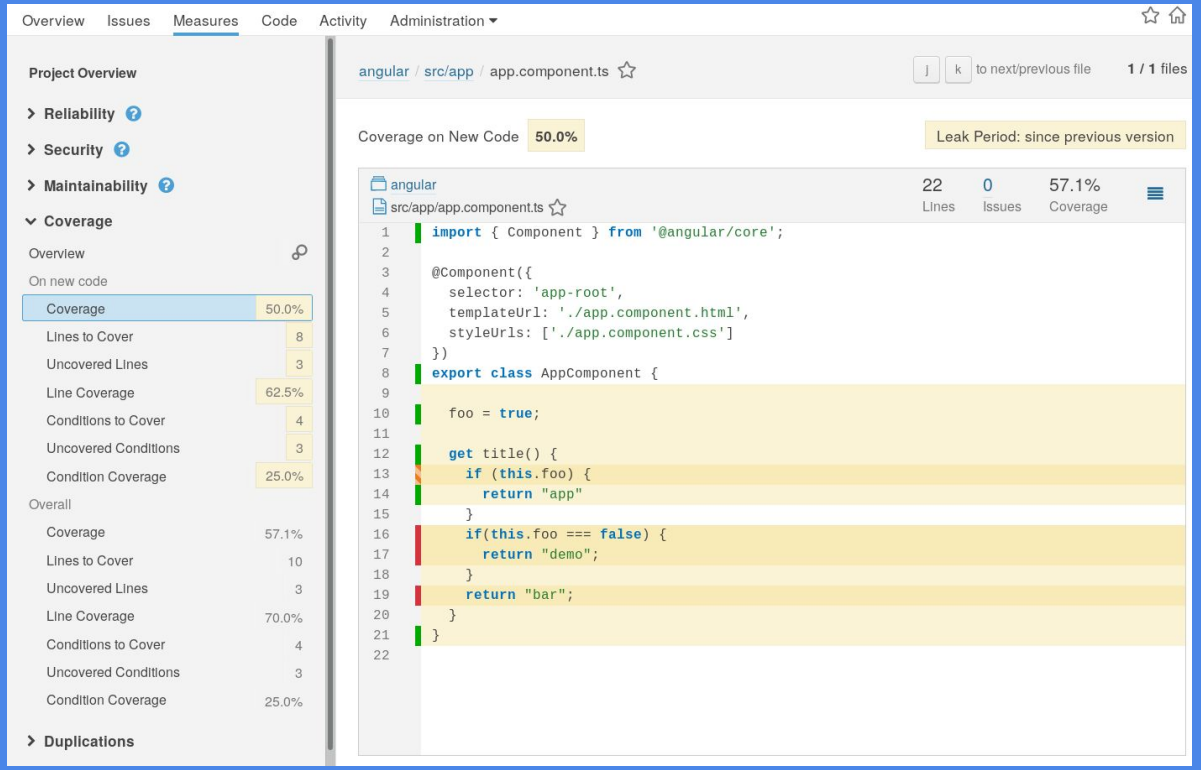
TypeScript

- Statische Analyse
  - IDE

```
1
2  /**
3   * A simple component providing an input field, an output area and a togg
4   */
5  @Component({
6      selector: "my-app",
7      template: `
8          <h1>Hello Angular 2!</h1>
9          <div>
10             <label [hidden]="!isCalling">I am {{name}}.</label>
11             <div>
12                 <input [(ngModel)]="name" />
13                 <button (click)="toggleCalling()">toggle Calling</button>
14             <div>
15         </div>
16         `
17 })
18 export class SimpleComponent {
19     name = "here";
20     private isCalling = true;
21
22     private toggleCalling() {
23         this.isCalling = !this.isCalling;
24     }
25 }
26
```

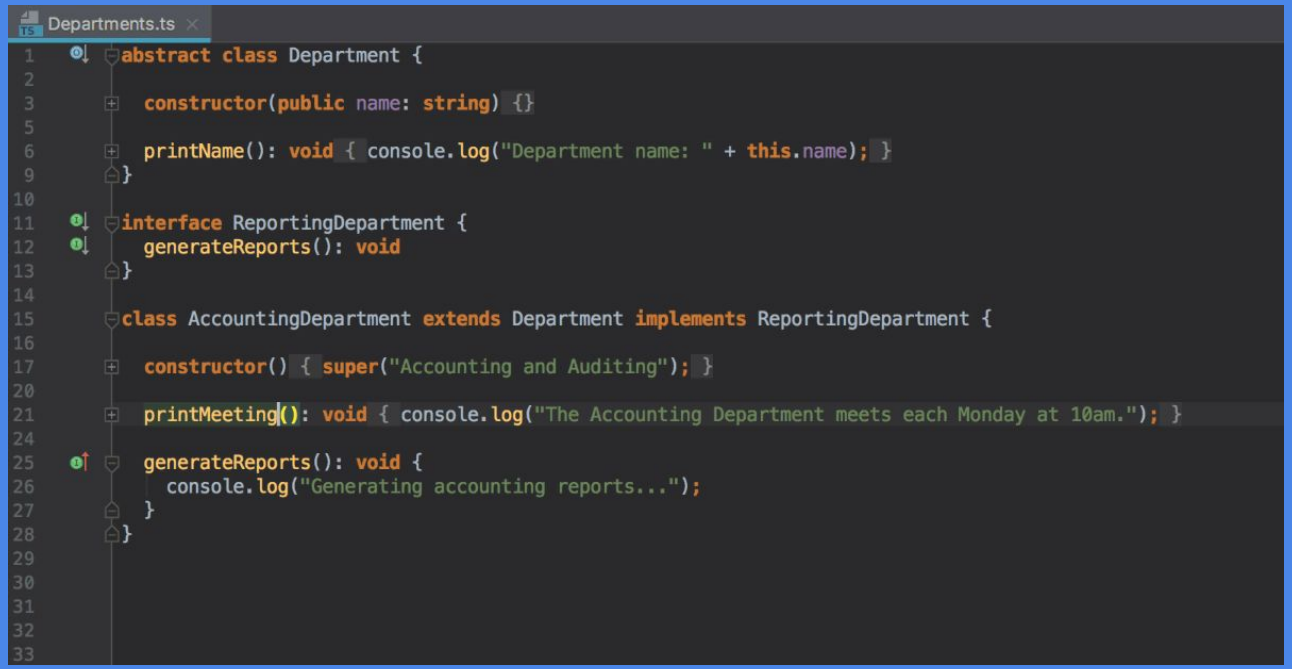
TypeScript

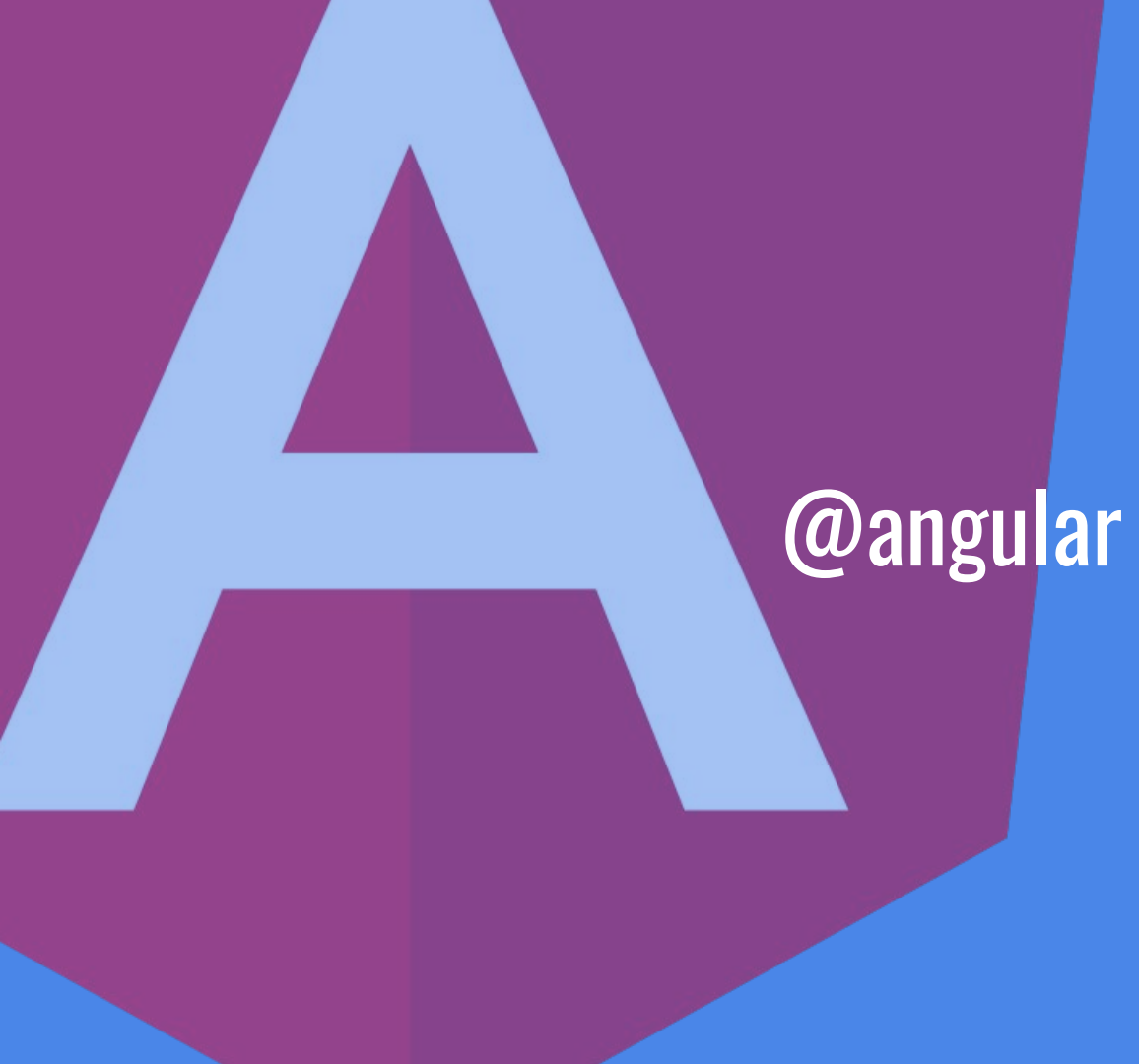
- Statische Analyse
  - IDE
  - Linter
  - SonarQube



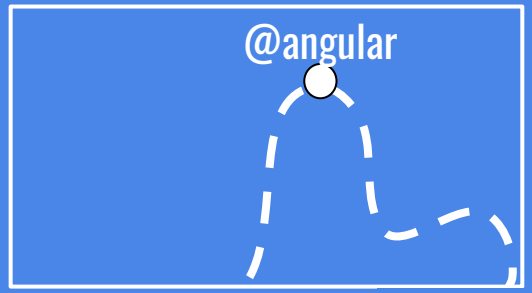
TypeScript

- Refactoring

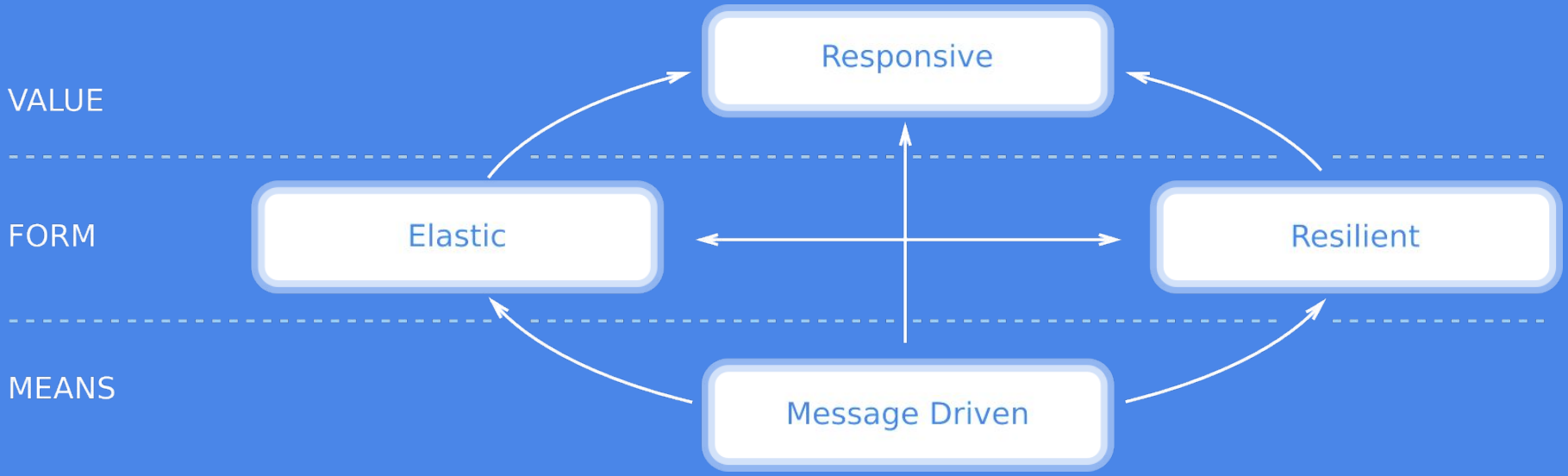



@angular

@angular

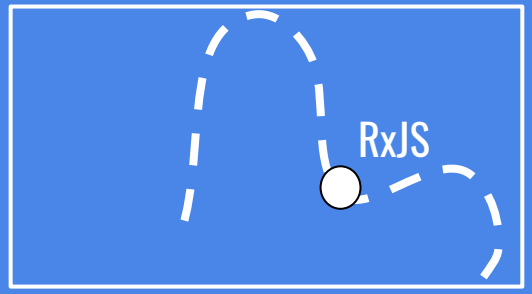


- Vorsicht: Veraltete Dokumentationen/Tipps
  - Datum beachten (Stackoverflow, ...)
  - Bücher altern schnell
- Tutorials von angular.io nutzen
- Austausch mit Community
  - User Groups, Meetups, Vorträge
- Auf dem laufenden bleiben
  - Aktuelle Magazine/Blogs


VALUE
FORM
MEANS
Responsive
Elastic
Resilient
Message Driven

Master RxJS

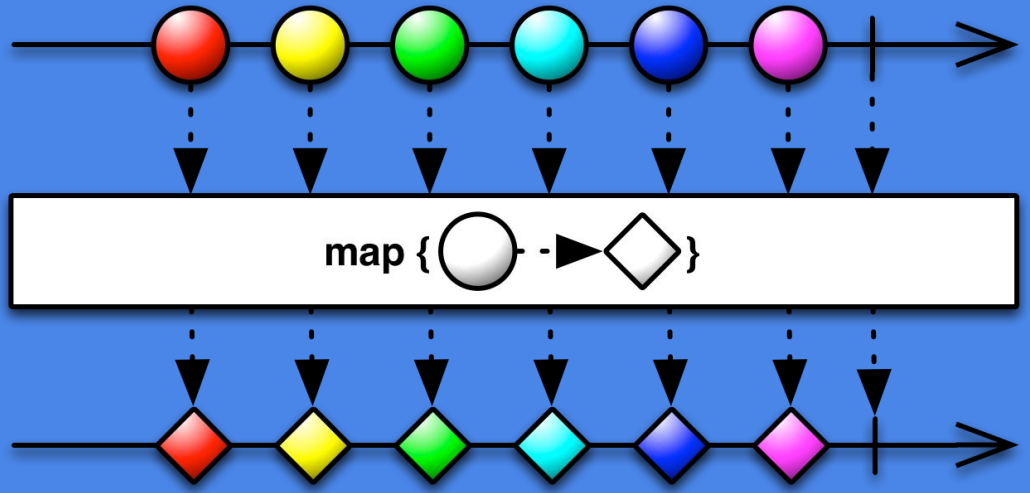
RxJS einsetzen



- RxJS - Programmieren mit **asynchronen Datenströmen**
- Browser arbeitet asynchron
- Von Angular intern verwendet
  - HTTP
  - Router
  - Forms
  - (ngrx)

RxJS lernen

- Verständnis: Funktionale Programmierung
  - “Filter-Map-Reduce”
- Rx-Operatoren kennenlernen
  - Marble-Diagramme



RxJS einsetzen

- Von Angular bereitgestellte API annehmen
  - Observables **nicht** direkt auflösen
    ~~“Subscribe und ab in die synchrone Welt”~~
  - Meist ist asynchrone Abhandlung sinnvoller
- `async`-Pipe!
  - Auflösen der Observables im Template
  - Angular erledigt subscribe/unsubscribe automatisch

Rxjs Anti Patterns

- Unsubscribe-Leak
  - Unsubscribe vergessen - Memory Leaks
  - Abhilfe: **`async`**-Pipe, **`ng2-rx-componentdestroyed`**

```
@Component({...})
export class MyComponent implements OnInit, OnDestroy
{
  ngOnInit() {
    Observable.interval(1000)
        .pipe(untilComponentDestroyed(this) )
        .subscribe(console.log);
  }
  ngOnDestroy() {}
}
```

Rxjs Anti Patterns

- **`async`**-Pipe - Kann zu mehrfachen (ungewollten) Requests führen, daher:
  - share()-Operator verwenden
  - @ngrx nutzen
  - ngIfAs-Syntax nutzen

```
<p *ngIf="userObservable | async as user">
  {{user.nachname}}, {{user.nachname}}
</p>
```

Rxjs Anti Patterns

- Nested Subscriptions

```
this.http.get(`/api/user/42`)
  .map( user => user.id)
  .subscribe( id => {

    this.http.get(`/api/shoppingListForUser/&{id}`)
      .subscribe(list => {
        this.shoppingList = list
      });

  });
```

Rxjs Anti Patterns

- Nested Subscriptions

```
this.http.get(`/api/user/42`)
  .map( user => user.id)
  .subscribe( id => {

    this.http.get( /api/shoppingListForUser/&{id}`)
      .subscribe(list => {
        this.shoppingList = list
      });

  });
```

Rxjs Anti Patterns

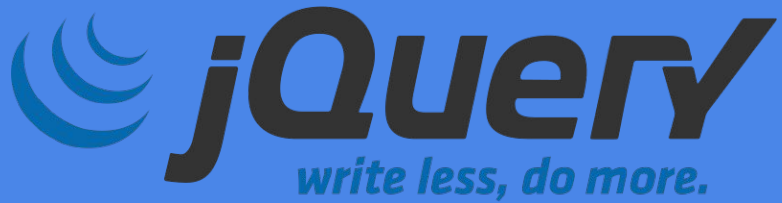
- mergeMap statt Nested Subscriptions

```
this.http.get(`/api/user/42`)
  .map( user => user.id)
  .mergeMap( id => {
    return this.http.get(`/api/shoppingListForUser/&{id}`)
  })
  .subscribe(list => {
    this.shoppingList = list
  });
```

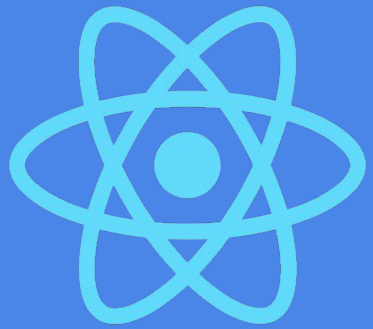
Rxjs Anti Patterns

- Nicht Promise-API und Rx-API mischen

- Bei Fremdbibliotheken
  - Mappingschicht
  - .toPromise() / .fromPromise()

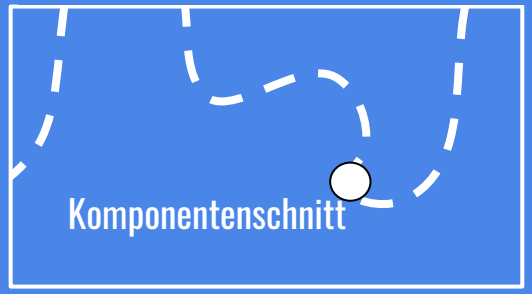


Umsteigen im Kopf



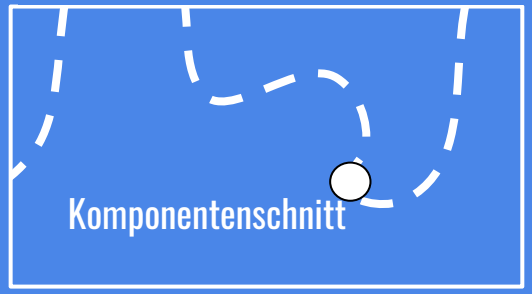
Anwendungs Architektur

Domain vs. View Components



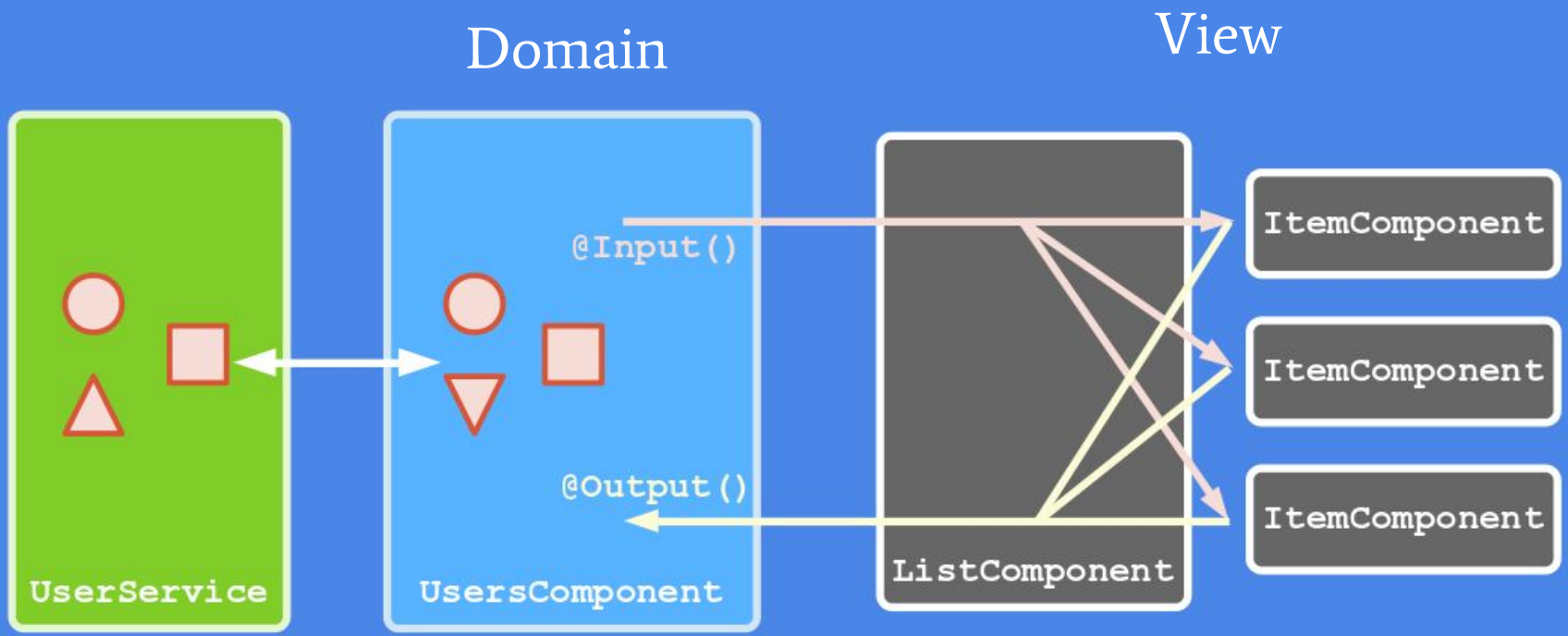
- View Komponenten
  - User-Interaktionselemente (Dropdown, Datepicker, Modal, ...)
  - **Hohe** Wiederverwendbarkeit
  - Zustandslos
  - Auslagerung als Option
    - Eigene Komponentenbibliothek

Domain vs. View Components



- Domain Komponenten
  - Komplexere und spezifischere (fachliche) Logik
  - Orchestrieren View-Komponenten
  - Mappen Datenmodell für View-Komponenten
  - Anbindung an Store, fachliche Services etc.
  - Möglicherweise zustandsbehaftet

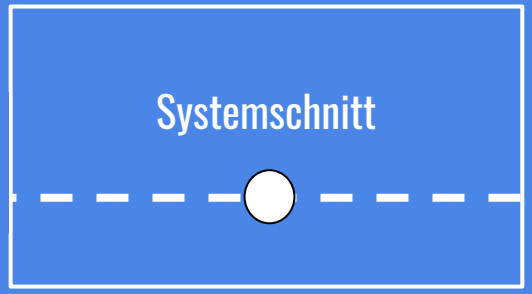
Domain vs. View Components

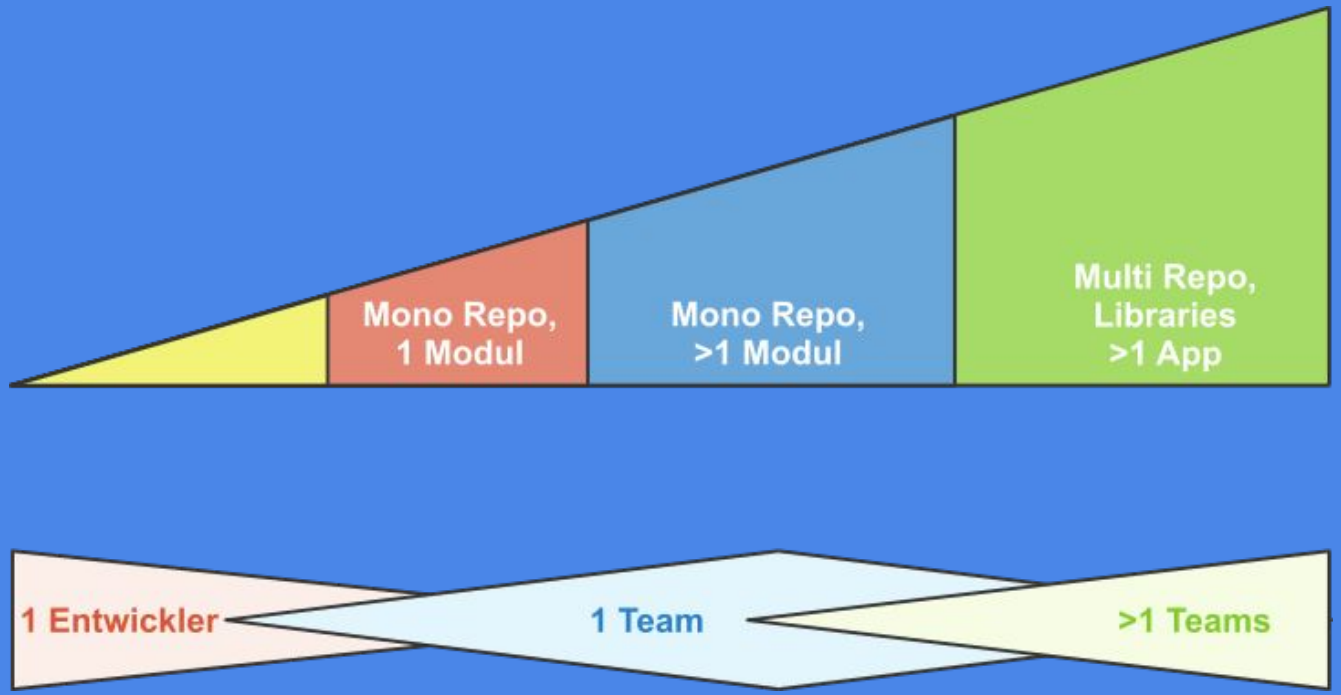


Komponenten-Frameworks

- Einsatz fertiger Komponenten-Frameworks
  - Material, Bootstrap, ...
  - Kopplung an Framework bewerten
- Styling durch CSS
  - Nur View-Components stylen!
  - CSS Architektur ist Thema für sich ...

Anwendungsschnitt





Anwendungsschnitt

- Anwendung nach Features aufbauen
  - Sowohl *technische* als auch *fachliche* Features
- Kapselung der Features in Module (`@NgModule()`)
  - Lazy Loading und Code-Splitting
- Option: Separierung in mehrere Anwendungen
  - “Self Contained Systems”
- Bereitstellung von Modulen via NPM Artefakt-Repository

Wiederverwendbare Module

- Vorgehen
  - Schnitt nach Einsatzzweck
  - Angular Package Format (APF) verwenden (entsprechend momentaner Angular Version)
  - Wird für volle Unterstützung (etwa AoT) benötigt
- Tools, die APF-konforme Packages generieren
  - ng-packagr
  - Angular CLI ab Version 6

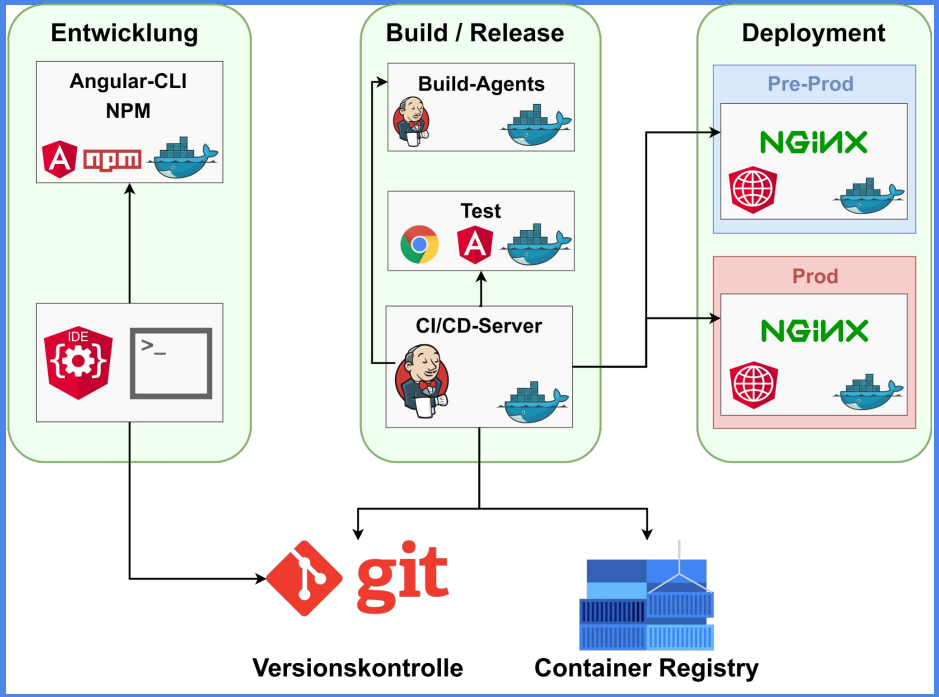
Build/Deployment

Docker

- Für Build
  - Umgebung benötigt keine speziellen Werkzeuge
- Für Dev
  - Bereitstellung von Backend/Frontend Services
- Für Tests
  - Isolierter Test, kleinerer Test-Scope
  - Mit Docker eigener Datenbestand simpel
  - Reproduzierbare, stabile Tests
- Für Deployment
  - Atomar, rollbackfähig, schrittweiser Rollout

Docker

- hub.docker.com/r/trion/
- trion/ng-cli
  - Angular-CLI, node, npm, yarn
- trion/ng-cli-karma
  - Chrome Browser, xvfb
- trion/ng-cli-e2e
  - Java, webdriver
- Docs
  - Englisch, Deutsch



Testing

Tests automatisieren

- Unit- und E2E-Tests
- Sollten automatisiert in der CI/CD Pipeline laufen
  - Inkl. Code-Coverage
  - `ng test -cc` bzw. `ng test --code-coverage`

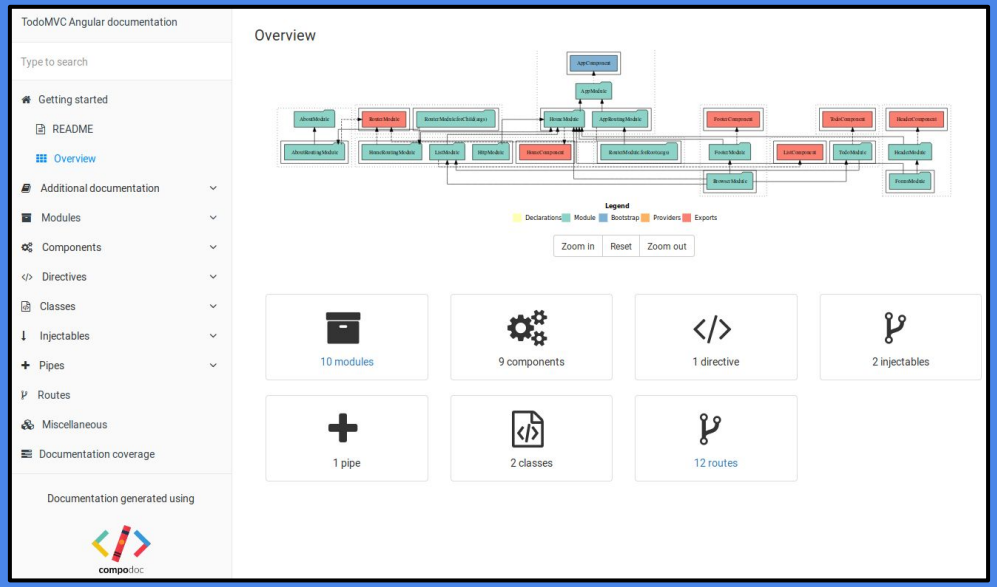
passed	#405 x86_64 docker	lint	01:21 5 minutes ago	
passed	#406 x86_64 docker	test:karma	01:27 5 minutes ago	92.86%

Code Qualität

- Code Coverage
- Linting (tslint/codelyzer)
  - Ebenfalls Teil der CI
  - `ng lint`

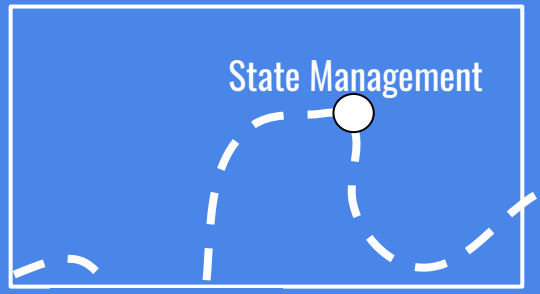
- Quellcode-Dokumentation
  - *@compodoc*
  - Doc-Coverage

@compodoc



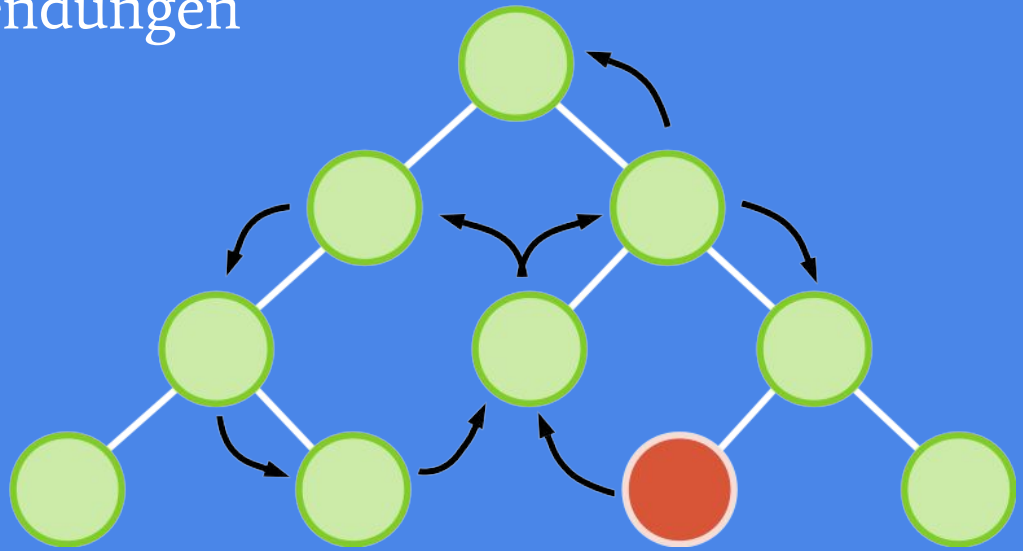
State Management

State Management



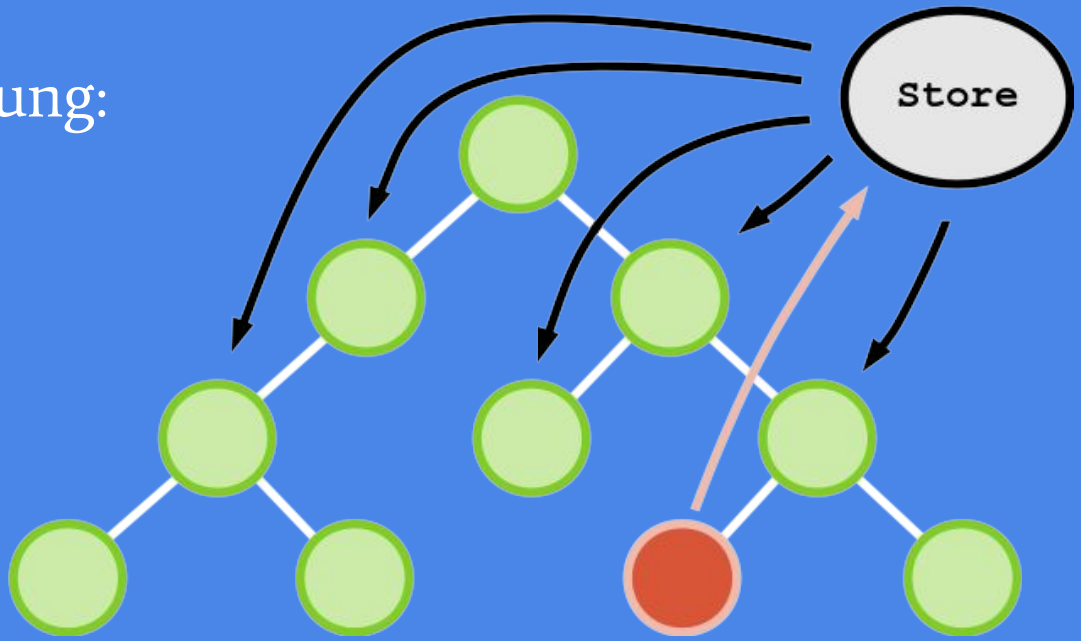
- Wo wird Zustand der Anwendung gehalten?
  - Komponenten, Services, Template, ...?
- Große/Komplexe Anwendungen

Komponente, von der die Änderung ausgeht



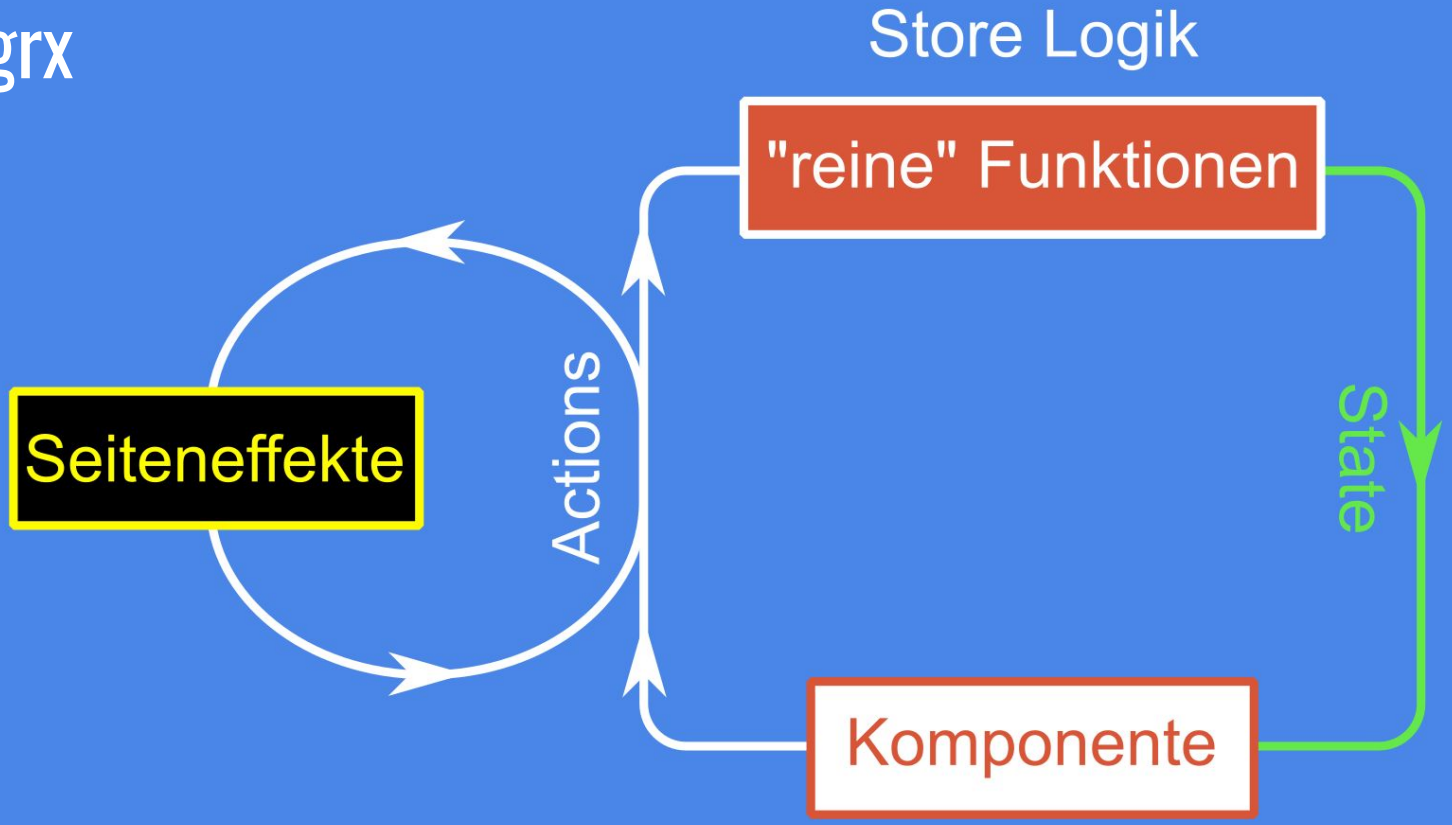
Flux/Redux

- Wo wird Zustand der Anwendung gehalten?
  - Zentraler Store
- Eine mögliche Ausprägung:
  `@ngrx`



Komponente, von der die Änderung ausgeht

@ngrx


Store Logik
"reine" Funktionen
Actions
State
Seiteneffekte
Komponente

Update Policy

“We (Google) also have one version policy, which means we are not allowed to keep existing applications on old versions of Angular”

Ng-conf Day 1 Keynote 2018

ng update

Fazit

- Angular stellt eine Option mit vielen Vorteilen dar
- Das Umfeld entwickelt sich kontinuierlich weiter
- Adäquate Pflege der Anwendungen und Entwickler unerläßlich
  - Kontinuierliche Fortbildung
  - Code-Review (Team-Intern/Extern)
  - Werkzeuge sinnvoll einsetzen

https://sitterberg.com/
karsten@sitterberg.com
@kakulty