



Angular Workshop

© it-economics 2018 | Alexander Lechner



Agenda

Basics

» NodeJs

- Installation
- NPM

» Typescript

- Typen
- Funktionen
- Interfaces
- Klassen
- Module

Advanced

» Angular

- Komponenten
- Formulare
- Services
- HTTP-Calls
- Unit-Tests
- Routing
- Module
- Libraries publishen
- Internationalisierung

Unterlagen

Zum Download

» <http://bit.ly/2B1JDeN>

Handout und Beispielcode inkl. Windows-Dependencies

» <https://github.com/it-economics/angular-example>

Beispielcode

JavaScript

... hat nichts mit Java zu tun

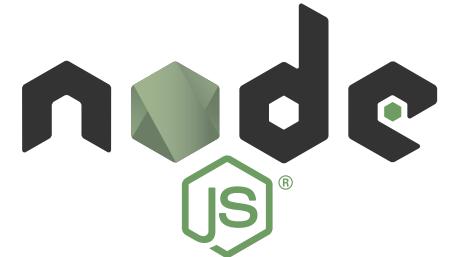
- » 1995 wird bei Netscape aus Mocha LiveScript und am Ende JavaScript
- » Brendan Eich gilt als Erfinder von JavaScript
- » 1996 entwickelt Microsoft Jscript
- » 1996 wurde JavaScript an die European Computer Manufacturers Association (ECMA) übergeben
 - JavaScript Standard ECMAScript ab 1997
 - Aktuelle Version des Standards: ECMAScript 2018 (9. Version)
- » JavaScript ist
 - imperativ und strukturiert
 - dynamisch typisiert
 - objektorientiert
 - funktional



NodeJs

JavaScript everywhere

- » JavaScript-Engine auf Basis von Chromes JavaScript-Engine V8
- » Erfinder: Ryan Dahl
- » Event-Driven Architektur
- » Asynchronous/non-blocking I/O
- » single-threaded event-loop basierend auf der C-Library libuv



NPM

Package Manager für JavaScript

- » Online-Datenbank für alle JavaScript Packages
- » Teil der NodeJs-Installation
- » Weltweit größte Sammlung von Open-Source-Bibliotheken, Anwendungen und Modulen
- » Sonatype Nexus und Artifactory unterstützen ebenfalls die NPM-API
- » Sinopia oder Verdaccio können lokal als NPM-Ersatz/-Cache genutzt werden



Sinopia: <https://www.npmjs.com/package/sinopia>
Verdaccio: <https://github.com/verdaccio/verdaccio>

NodeJs Installation

- » Variante 1: Download von der NodeJs-Homepage
- » **Variante 2:** Installation über den **Node Version Manager**
- » NVM für Linux & Mac: <https://github.com/creationix/nvm>
- » NVM für Windows: <https://github.com/coreybutler/nvm-windows>

```
nvm
```

```
nvm ls-remote
```

```
nvm install v10.13.0
```

```
node -v
```

NodeJs

Proxy konfigurieren

» Im Corporate Network kann es nötig sein einen Proxy zu konfigurieren:

```
# Proxy für HTTP-Verbindungen
npm config set proxy http://<username>:<password>@<proxy-server-url>:<port>
```

```
# Proxy für HTTPS-Verbindungen
npm config set https-proxy http://<username>:<password>@<proxy-server-url>:<port>
```

```
# HTTP-Verbindung zur Registry statt HTTPS
npm config set registry http://registry.npmjs.org
```

```
# Alternative
npm config edit
```

Kata

FizzBuzz

Schreibe eine Funktion, die die Zahlen von 1 bis 100 zurückgibt.

Manche Zahlen sollen dabei allerdings übersetzt werden:

- Für Vielfache von 3 gibt „Fizz“ aus.
- Für Vielfache von 5 gibt „Buzz“ aus.
- Für Vielfache von 3 und 5 gibt „FizzBuzz“ aus.

Unit-Testing-Library: **Jasmine** (<https://jasmine.github.io/>)

```
npm init
npm install --save-dev jasmine
node node_modules/jasmine/bin/jasmine.js init
```

» Dependencies werden in der `package.json` definiert

» Es gibt drei Möglichkeiten Abhängigkeiten anzugeben

- **dependencies:**

- Javascript-Objekt - ordnet NPM-Packages einer Versionsrange zu
- statt einer Versionsrange kann auch der Speicherort des Packages oder das Git-Repository angegeben werden
- Abhängigkeiten werden im Ordner `node_modules` gespeichert
- Jedes Package hat seinen eigenen `node_modules`-Ordner

- **devDependencies:**

- Gleicher Aufbau wie `dependencies`
- Enthält Abhängigkeiten für Testing, Compilation oder Transpiling, die Nutzer des Projektes nicht benötigen

- **peerDependencies:**

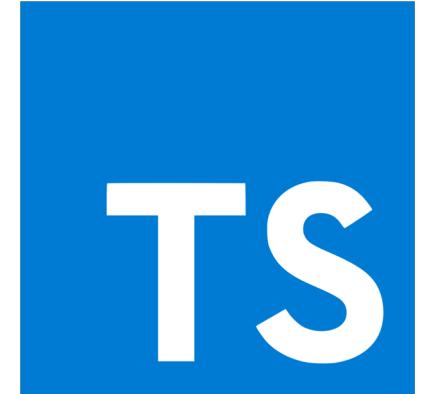
- Ebenfalls gleicher Aufbau wie `dependencies`
- Jede Abhängigkeit hat eigene Abhängigkeiten – es kann vorkommen dass ein Package in mehreren Versionen im Dependency-Tree enthalten ist.
- Sind diese Versionen inkompatibel kann es zu Runtime-Fehlern kommen
- Mit `peerDependencies` kann man die Kompatibilität des Projektes mit einer bestimmten Dependency-Version klar definieren
- Wird ein Interface einer Dependency durch das Projekt angeboten (explizit oder implizit) sollte man die Abhängigkeit als peer dependency angeben.

TypeScript

Microsofts typed superset of JavaScript

- » „Typescript is a typed superset of JavaScript that compiles to plain JavaScript.“
 - Kompiliert standardmäßig in ECMAScript 3
 - Gültiger JavaScript-Code ist auch gültiger TypeScript-Code
- » 2012 in der Version 0.8 veröffentlicht
 - IDE-Unterstützung nur durch Microsoft Visual Studio (nur Windows)
 - 2013 erstes Plug-In für Eclipse
- » Typescript 1.0 wurde 2014 veröffentlicht
- » Alle 2-3 Monate kommt eine neue Version
- » Aktuelle Version: 3.1
- » Viele Punkte aus TypeScript wurden in ECMAScript 6 übernommen

TypeScript



TypeScript

Basistypen

» boolean

» number

» string

» array

» tuple

» enum

» any

» void

» null & undefined

» never

» object

» TypeScript Handbook:

» <http://www.typescriptlang.org/docs/home.html>

TypeScript

Interfaces - duck-typing

- » Typescript arbeitet nach dem Prinzip des „duck-typing“ oder „structural subtyping“
- » Beschreibt wie ein Objekt aussieht
- » Explizite Nennung des Interfaces ist möglich, aber kein Muss
- » Können auch Funktionstypen beschreiben
- » Kann von Klassen erben

```
interface LabelledValue {  
    label: string;  
}  
  
function printLabel(labelledObj: LabelledValue)  
{  
    console.log(labelledObj.label);  
}  
  
let myObj = {  
    size: 10,  
    label: "Size 10 Object"  
};  
printLabel(myObj);
```

TypeScript

Klassen

- » Modifier
 - public
 - private
 - protected
 - readonly
 - static
 - abstract
- » Properties sind public by default

```
class Animal {  
  
    name: string;  
  
    constructor(theName: string) {  
        this.name = theName;  
    }  
  
    move(distanceInMeters: number = 0) {  
        console.log(  
            `${this.name} moved${distanceInMeters}m.`  
        );  
    }  
}
```

TypeScript

Kontrollstrukturen

for-loop

```
let list = [4, 5, 6];

for (let i in list) {
  console.log(i); // "0", "1", "2",
}

for (let i of list) {
  console.log(i); // "4", "5", "6"
}
```

if-else

```
if(x) {
  console.log('fizz');
} else {
  console.log('buzz');
}
```

TypeScript

Module

- » Klassen und Interfaces sind standardmäßig nicht außerhalb ihres Moduls bzw. ihres Files sichtbar
- » Was sichtbar sein soll muss exportiert werden
- » Exportierte Klassen, Interfaces, Funktionen, etc. können in anderen Modulen importiert werden
- » Jedes Modul kann genau einen **default**-Export definieren

```
export interface StringValidator {  
    isAcceptable(s: string): boolean;  
}  
  
// renaming  
class ZipCodeValidator implements StringValidator {  
    isAcceptable(s: string) {  
        return s.length === 5 && numberRegexp.test(s);  
    }  
}  
  
export {ZipCodeValidator};  
  
// Imports  
import {ZipCodeValidator} from './ZipCodeValidator';
```

Kata

RingBuffer

- » Entwickle eine Klasse, die einen Ringbuffer [1, 2] implementiert.
- » An einen Ringbuffer können Werte „hinten“ angehängt werden wie an eine Queue (*add()*).
- » Und sie können „vorne“ entnommen werden wie aus einer Queue (*take()*).
- » Allerdings hat der Ringbuffer eine begrenzte Kapazität (*size()*). Wenn die ausgeschöpft ist, werden Werte „am Anfang“ „überschrieben“, d.h. dann haben neue Werte Vorrang vor alten.
- » Das Interface der Klasse soll so aussehen →

<https://ccd-school.de/coding-dojo/class-katas/ringpuffer/>
TS-Dojo Boilerplate: <https://github.com/alxlchnr/ts-dojo-boilerplate>

```
class Ringbuffer<T> {  
  
    constructor( private _size: number ) {}  
  
    add(value: T): void {...}  
  
    take(): T {...}  
  
    count(): number {...}  
    // Anzahl ungelesene Elemente (<= Size())  
  
    get size(): number {...} // Größe des Ringbuffers  
}
```

Angular

Just „Angular“

- » angular (engl. eckig) bezieht sich auf < > in HTML
- » 2010 Release von AngularJs
- » Erfinder: Misko Hevery (Google)
- » 2014/2015 Rewrite
- » 2016 Release von Angular 2.0
- » Aktuelle Version Angular 7



Angular

Ein neues Projekt aufsetzen

Angular-Projekt mit Hilfe von Angular-CLI erstellen

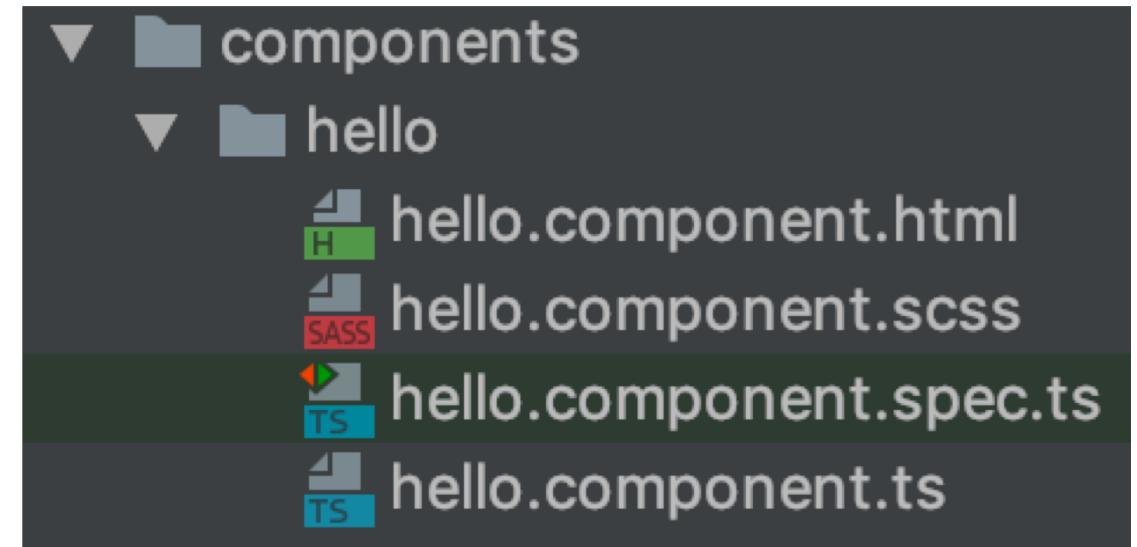
```
npm install -g @angular/cli  
  
ng new example-project  
  
cd example-project  
  
npm install
```

Für den Style: Bootstrap hinzufügen

```
npm install bootstrap --save  
  
npm install ngx-bootstrap --save  
  
ng add ngx-bootstrap  
  
# install peer dependencies if necessary  
npm i popper.js@^1.14.3 --save  
  
npm i jquery@1.9.1 --save
```

Angular Komponenten

- » `ng generate component components/hello`
- » Komponente beschreibt einen Teil des sichtbaren Bildschirms ~ View
- » Eine Komponente besteht aus:
 - HTML-Template
 - Stylesheet (CSS welches hier definiert wird gilt nur für die Komponente)
 - Typescript-Klasse
 - Unit-Test
- » Properties und Methoden aus der Typescript-Klassen können im HTML-Template angezeigt bzw. genutzt werden
- » Neue Komponenten müssen der Anwendung bzw. dem App-Modul erst bekannt gemacht werden



```

@NgModule({
  declarations: [
    HelloComponent,
  ]
})

```

Angular

Template-Syntax

Databinding

```
Hallo {{ name }}!
```

Input

```
<app-my-component [inputProp]="<value>">
```

Kontrollstrukturen

*ngIf

```
<p *ngIf="name">  
  Hallo {{ name }}!  
</p>  
<p *ngIf="!name">  
  Hallo!  
</p>
```

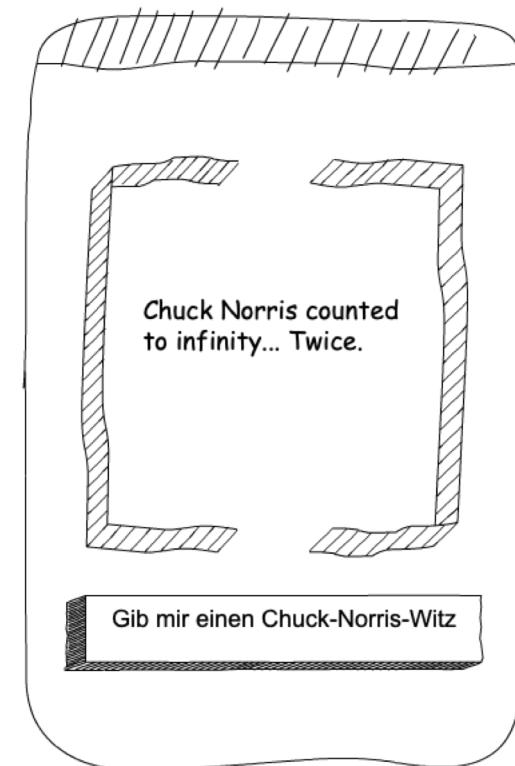
*ngFor

```
<ul>  
  <li *ngFor="let name of names">  
    {{ name }}  
  </li>  
</ul>
```

HandsOn

Aufgabe 1

- » Am Ende des Workshops werden wir eine Angular-Anwendung haben, die randomisiert Chuck-Norris-Witze von einer öffentlichen API abholen und anzeigen kann.
- » 1. Aufgabe:
 - Erstelle ein Angular-Projekt.
 - Die Angular-App zeigt einen Button „Gib mir einen Chuck-Norris-Witz“.
 - Bei einem Klick auf diesen Button wird immer derselbe Chuck-Norris-Witz über dem Button angezeigt.
 - *Fleißaufgabe:* Es soll einigermaßen ansprechend aussehen.



Angular Formulare

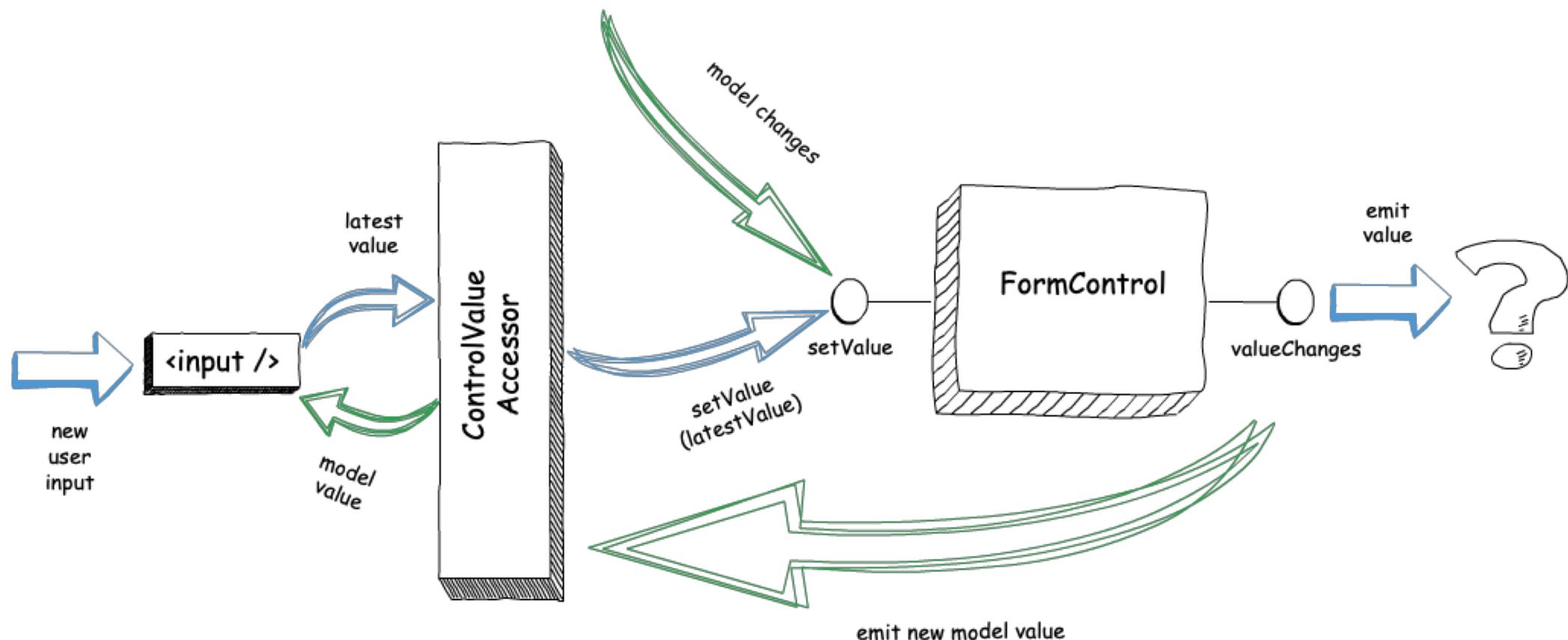
- » Ein Formular in Angular besteht aus FormControl
- » FormControl tracken den Wert und den Validierungsstatus eines Eingabefeldes
- » FormControl können zu FormGroup zusammengefasst werden
- » ControlValueAccessors dienen als Bridge zwischen FormControl und dem entsprechenden DOM-Elementen
- » Es gibt zwei Formulartypen:
 - template-driven forms
 - reactive forms

Angular reactive forms

- » robust
- » skalierbar

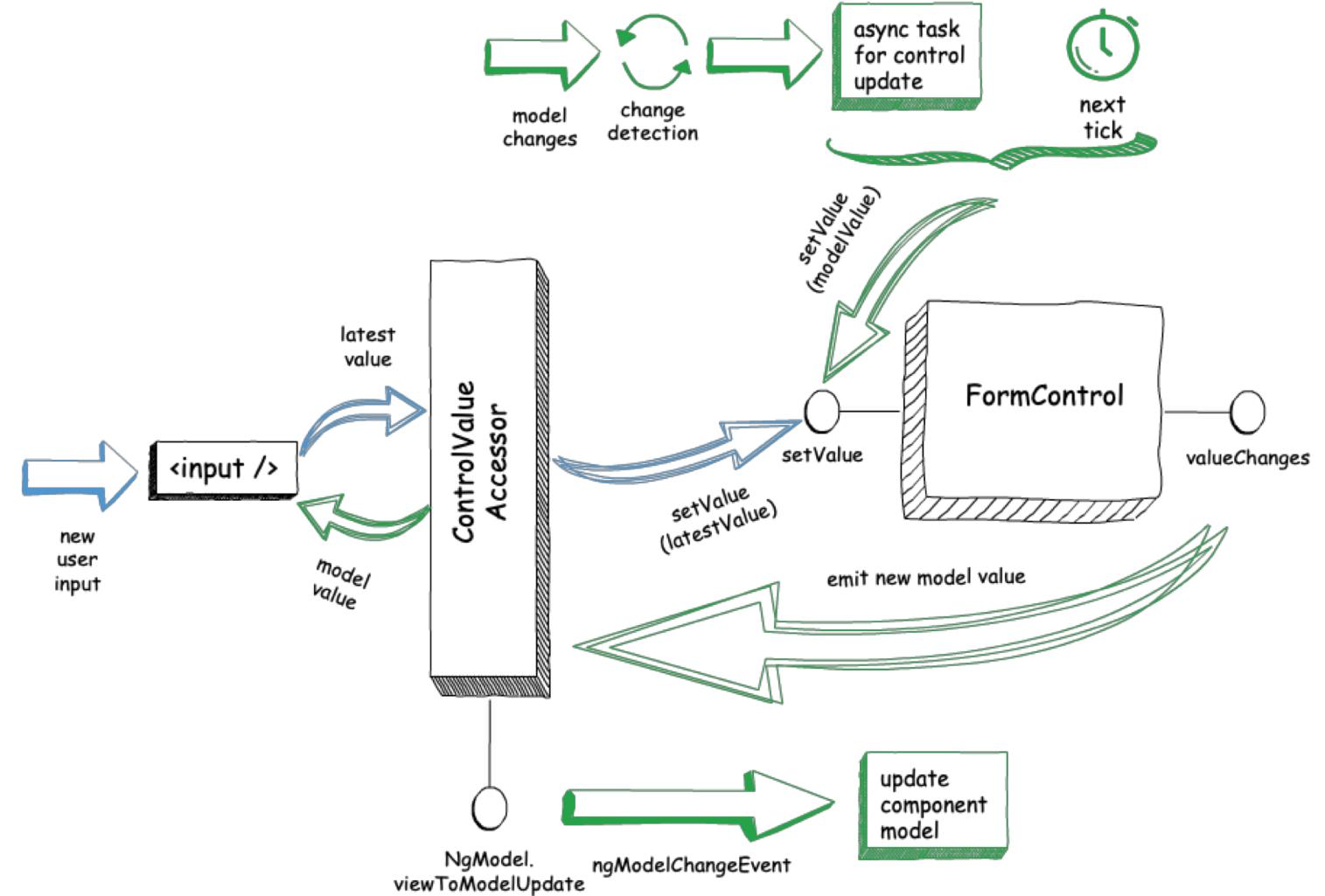
- » wiederverwendbar
- » testbar

- » Einsatzszenario: Formulare stellen die Hauptfunktionalität der Anwendung dar.



Angular template-driven forms

- » für einfache Formulare geeignet
- » einfach einzubinden
- » Einsatzszenario: Formularlogik ist ausschließlich im Template abbildbar.



Angular

template-driven forms - Validierung

- » Für Standardattribute von <input>-Tags, wie z.B. **required** oder **maxlength** gibt es passende Angular-Validatoren.
- » Bei jeder Eingabe werden die Validatoren ausgeführt.
- » Auftretende Validierungsfehler werden an das FormControl angehängt.
- » Es ist möglich eigene Validatoren zu schreiben
 - Direktive erzeugen
`ng generate directive directives/forbiddenNameValidator`
 - Validator-Interface implementieren
 - Direktive als Validator bekannt machen
- » Validatoren können auch auf FormGroup angewendet werden
- » Bei *reactive forms* sind keine Validierungsdirektiven nötig. Die eigentliche Validierungslogik wird dann in eine **ValidatorFn** ausgelagert.

```
<input
  type="text" class="form-control"
  [(ngModel)]="userName"
  name="userName"
  appForbiddenNameValidator="Alex"
  required
/>
```

```
export interface Validator {
  validate(control: AbstractControl): ValidationErrors | null;
  registerOnValidatorChange?(fn: () => void): void;
}
```

Angular

HttpClient-Service nutzen

- » Angular bietet eine Vielzahl von Services die man sofort benutzen kann, z.B. HttpClient
- » Angular nutzt *constructor based dependency injection*
- » Der HttpClient bietet für jede HTTP-Operation entsprechende Methoden
- » Jede dieser Methoden liefert ein Observable zurück
- » Angular benutzt RxJs (Reactive Extension for JavaScript) für jeglichen asynchronen Code

```
import {HttpClientModule} from  
'@angular/common/http';  
// ...  
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  HttpClientModule  
,  
// ...
```

```
constructor(private httpClient: HttpClient){  
}
```

Exkurs

Reactive Programming

- » Legt den Fokus auf Events statt auf einen State
- » Wichtigste Funktionen für diesen Workshop:
 - **of** – Erzeugen von Observables
 - **pipe** – bearbeitet die Werte/Events in einem Stream
 - **map**
 - **tap**
 - **filter**
 - **subscribe**
- » <https://angular.io/guide/rx-library>

“Reactive programming is programming with asynchronous data streams. On top of that, you are given an amazing toolbox of functions to combine, create and filter any of those streams.”

[staltz/introrx.md](#)

Angular Services

- » `ng generate service services/weather`
- » Um einen Service z.B. in einer Komponente injecten zu können muss er einem Injector hinzugefügt werden
 - Über einen Provider wird ein Service einem Injector hinzugefügt
 - `@Injectable({ providedIn: 'root' })` macht einen Service dem 'root'-Injector bekannt.
 - Provider können in der `@NgModule`-Annotation oder der `@Component`-Annotation definiert werden
- » Services sind Singletons im Scope ihres Injectors
- » Injector sind hierarchisch aufgebaut
 - Niedrigste Ebene: Component-Injector
 - Module-Injector
 - Root-Injector
 - Auf der Suche nach einer Serviceinstanz wird der Baum von unten nach oben durchwandert

```
@NgModule({
  // ...
  Providers:[
    {provide: XYService, useClass: XYServiceImpl},
    {provide: XYService, useValue:
      xYServiceInstance},
    {provide: OldLogger, useExisting: NewLogger}
  ]
})

// ProviderFactory
let xYServiceProvider =
  (dep1: Dependency1, dep2: Dependency2) =>{
    return new XYServiceImpl(dep1, dep2);
  }
// ...
{
  provide: XYService,
  useFactory: xYServiceProvider,
  deps: [Dependency1, Dependency2]
}
```

HandsOn

Aufgabe 2

- » Es gibt eine öffentliche API für Chuck-Norris-Witze: <https://api.icndb.com>
- » Dazu passende Website: <http://www.icndb.com/api/>
- » 2. Aufgabe:
 - Beim Klick auf den Button „Gib mir einen Chuck-Norris-Witz“ wird von der Chuck-Norris-API jedes Mal ein neuer Witz geladen und angezeigt.
 - Es gibt einen Angular-ChuckNorrisJoke-Service

HandsOn

Aufgabe 3

» 3. Aufgabe: Schreibe Unit-Tests für die Chuck-Norris-App.

Angular

HttpInterceptors

- » HttpInterceptors können verwendet werden um
 - Querschnittliche Funktionalität bei jedem HTTP-Call zu implementieren
 - Credentials an Request anhängen
 - Caching zu implementieren
 - Ladeanimationen ein und auszublenden.
- » Die Reihenfolger der Interceptors ist abhängig von der Reihenfolge in der sie provided werden
- » Die Objekte der Klassen HttpRequest und HttpResponse sind immutable. Um Änderungen an einem Request durchzuführen müssen die Objekte geklont werden.

```

@Injectable()
export class AppIdInterceptor implements
HttpInterceptor {

  intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {

    const newReq = req.clone(
      { url: req.url + '&APPID=<AppId-Value>' }
    );
    return next.handle(newReq);

  }
}

// @NgModule
{ provide: HTTP_INTERCEPTORS, useClass:
  AppIdInterceptor, multi: true }

```

Angular

Mit Modulen den Überblick behalten

- » Um ein neues Modul zu erstellen
 - `ng generate module weather`
 - Komponenten, Direktiven, Services etc. in den Modul-Ordner verschieben
 - Komponenten und Direktiven dem neuen Modul bekannt machen
 - Öffentliche Komponenten im neuen Modul exportieren
 - Verschobene Komponenten etc. aus dem `declarations`-Attribut aus dem AppModul entfernen
 - Mit Providern das Spiel wiederholen
 - Neues Modul im AppModul als Import deklarieren
- » In vielen Angular-Anwendungen werden standardmäßig folgende Module angelegt:
 - **CoreModule**: enthält nur Singleton-Services die beim AppStart geladen werden sollen; sollte keine declarations enthalten
 - **SharedModule**: enthält Komponenten, Direktiven und Pipes, die in der ganzen Anwendung verwendet werden; sollte keine Provider enthalten
 - Mehrere **FeatureModules**

```
@NgModule({
  declarations: [ WeatherComponent ],
  imports: [
    CommonModule,
    HttpClientModule
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AppIdInterceptor,
      multi: true}
  ],
  exports: [ WeatherComponent ]
})
export class WeatherModule { }
```

Angular

Navigation aka Routing

- » Navigation von einer Komponente zu einer Anderen
- » Es müssen Routen definiert werden und dem **RouterModule** übergeben werden
- » Eine Route besteht aus
 - Einem Pfad
 - Einer Zielkomponente, einem Redirect-Pfad, oder einem ChildModul
 - Optional zusätzliche Daten
- » Bei der Navigation zwischen Seiten wird die Browser location und die Browser history angepasst
- » Bei der Navigation passiert kein Reload - um das in den Browsern zu erreichen gibt es zwei LocationStrategies zur Auswahl
 - **PathLocationStrategy** - "HTML5 pushState" style
 - **HashLocationStrategy** - "hash URL" style.
- » Die Routen werden meist in einem eigenen RoutingModul definiert

```
const routes: Routes = [
  {
    path: '',
    component: AppComponent,
    children: [
      {
        path: 'weather',
        component: WeatherComponent
      }, {
        path: 'home',
        component: HomeComponent
      }, {
        path: '',
        redirectTo: '/home',
        pathMatch: 'full'
      }
    ]
  ];
@NgModule({
  imports: [
    RouterModule.forRoot(routes, {enableTracing: true})
  ],
  exports: [RouterModule]
}) export class AppRoutingModule { }
```

Angular

Routing & Guards

- » Bei der Aktivierung der definierten Routen können Guards Bedingungen überprüfen
- » Mit Guards kann man
 - Bestimmte Routen nur angemeldeten Benutzern erlauben
 - Bestimmte Module nur bei bestimmten Bedingungen laden
 - Dynamisch zusätzliche Daten für die Route ermitteln
- » Es gibt folgende Guards:
 - CanActivate
 - CanActivateChild
 - CanDeactivate
 - Resolve
 - CanLoad
- » Guards werden bei der Routendefinition angegeben

```
@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
  constructor(
    private authService: AuthService,
    private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean> | Promise<boolean> | boolean {
    if ( !this.authService.isLoggedIn ) {
      this.router.navigate(
        ['/login'],
        {replaceUrl: true}
      );
    }
    return userIsLoggedIn;
  }
}
```

HandsOn

Aufgabe 4

» 4. Aufgabe:

- Füge einen Login-Dialog ein
- Nach erfolgreicher Anmeldung stehen dem Nutzer dieselben Funktionen zur Verfügung wie bisher
- Bei fehlerhafter Anmeldung wird eine Fehlermeldung angezeigt.
- Beim Login muss Benutzername und Passwort eingegeben werden
- Wenn das Passwort ‚secret‘ lautet wird der Nutzer authentifiziert.

Angular

Component Libraries erstellen und publishen

```
ng init
```

```
# Das Projekt kann sowohl Apps als auch Libraries  
# enthalten
```

```
ng generate application myapp  
ng generate library weather
```

```
ng serve --project myapp -app
```

```
ng build --project weather -lib
```

```
ng build --project weather --prod
```

```
cd projects/weather
```

```
npm publish
```

Angular

Bestehendes Modul als Library publishen

```
npm install ng-packagr --save-dev

# ng-package.json
{
  "$schema": "./node_modules/ng-packagr/ng-
              package.schema.json",
  "lib": {
    "entryFile": "public_api.ts"
  }
}

# public_api.ts
export * from './src/app/hello/hello.module'

# zusätzliches script in package.json
"packagr": "ng-packagr -p ng-package.json"
# dependencies sollten peerDependencies werden
```

```
npm run packagr

cd dist

# Projektname in dist/package.json ändern,
# denn der Name ist derselbe wie der Name
# der APP
npm publish
```

Angular Internationalisierung

```
<a i18n="<meaning>|<description>@@<id>">Hallo!</a>

ng xi18n --output-path locale

# AOT Prod Build
ng build --prod --i18n-file src/locale/messages.fr.xlf --i18n-format xlf --i18n-locale fr

# JIT Dev Build (main.ts)
const translations = require(`raw-loader!./locale/messages.fr.xlf`);

platformBrowserDynamic().bootstrapModule(AppModule, {
  providers: [
    {provide: TRANSLATIONS, useValue: translations},
    {provide: TRANSLATIONS_FORMAT, useValue: 'xlf'},
    {provide: LOCALE_ID, useValue: 'fr' }
  ]
});
```

Linksammlung

» NVM

- Linux & Mac: <https://github.com/creationix/nvm>
- Windows: <https://github.com/coreybutler/nvm-windows>

» Typescript DeepDive: <https://basarat.gitbooks.io/typescript/>

» Angular-Beispiel-Projekt: <https://github.com/it-economics/angular-example>

Feedback

» <http://bit.ly/2RMnXun>

it-economics

it is more than technology

Kontakt

Alexander Lechner

it-economics GmbH
Bothestr. 11, 2 OG
81675 München
www.it-economics.de

alechner@it-economics.de
Mobil +49 151 40 60 50 31



Best of Consulting 2014
1. Platz im Wettbewerb der WirtschaftsWoche.
Kategorie: IT-Management

Beste Berater 2014, 2015, 2017 und 2018 im Branchenreport von brand eins Wissen und Statista. Kategorie: IT-Strategie & IT-Implementierung

Great Place To Work 2016, Deutschland, ITK, Bayern