haake & partner
– datentechnik gmbh –

DHBW
Duale Hochschule
Baden-Württemberg

Clientseitige Webframeworks wie AngularJS, ReactJS und OpenUI5

Seminararbeit

für die Prüfung zum

Bachelor of Science (B.Sc.)

des Studiengangs Wirtschaftsinformatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

Verfasser	Sebastian Greulich, Fabio Krämer
Partnerunternehmen	
Matrikelnummer, Kurs	, WWI16B2
Wissenschaftlicher Betreuer	
Abgabe	17.09.2018

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit mit dem Thema: „*Clientseitige Webframeworks wie AngularJS, ReactJS und OpenUI5*“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, 17.09.2018

Sebastian Greulich, Fabio Krämer

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

2. Webframeworks

Beschreibung von Decoratoren

3. AngularJS

3.1. Allgemein

3.1.1. Entstehung des Frameworks

3.1.2. Entwicklungsumgebung

3.2. Konzepte

Im Folgenden werden die in AngularJS verwendeten Konzepte näher erläutert. Die Beispielanwendung in Abbildung 1 wird zur Erklärung der Konzepte verwendet. Bei Änderung des Namens im Input-Feld wird dieser in der obigen Überschrift auch verändert. Die Beispielanwendung weißt folgende Ordnerstruktur auf:

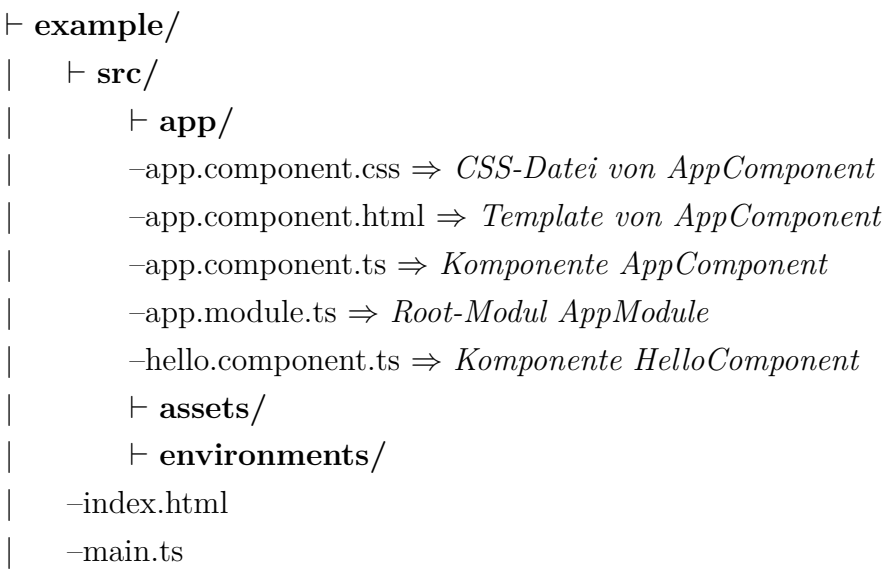

Abbildung 1.: Screenshot Angular-Beispielanwendung

```
|   –styles.css
|   –...
|   ⊢ node_modules/
|   ⊢ e2e/
| –...
```

Allgemeine Beschreibung von Metadaten und Klassen Dekorator.

3.2.1. Module

Eine Angular-Anwendung ist modular aufgebaut und kann demnach aus mehreren Modulen bestehen. Module fassen zusammengehörige Codeeinheiten zusammen und können bestimmte Funktionalitäten bereitstellen, die wiederum von anderen Modulen verwendet werden können. (vgl. Steyer et al. 2017, S. 103 ff.) Die Module einer Angular-Anwendung können in Root-Module, Feature-Module und Shared-Module unterteilt werden.

Den Begriff Bootstrapping erwähnen!

Wenn ein Client eine auf AngularJS basierte Seite anfordert, dann schickt der Server den Inhalt der *index.html* als Antwort an den Client zurück. Der Client führt daraufhin die im HTML-Dokument enthaltenen Skript-Elemente aus. Dabei wird die Angular-Plattform initialisiert und das Root-Modul übergeben. Dieses Root-Modul konfiguriert die Angular-Anwendung.(vgl. Steyer et al. 2017, S. 60; vgl. Freeman 2018, S. 226 ff.) Das Root-Modul der Beispielanwendung ist das Modul *AppModule* siehe Listing 3.1.

Module werden im Allgemeinen durch den Dekorator *@NgModule* gekennzeichnet und durch Eigenschaften konfiguriert. Ein Modul kann verschiedene weitere Module über die Eigenschaft *import* importieren und damit die bereitgestellten Funktionalitäten verwenden. Die vom Modul verwendeten Direktiven, Komponenten und Pipes werden in der Eigenschaft *declarations* angegeben. Jedes Root-Modul besitzt die Eigenschaft *bootstrap*. Diese Eigenschaft spezifiziert die Komponente, die beim Starten der Anwendung geladen werden soll.

Das Modul *AppModule* in Listing 3.1 importiert die Module *NgModule*, *BrowserModule* und *FormsModule* und deklariert die zugehörigen Komponenten *AppComponent* und *HelloComponent*. Beim Starten der Anwendung soll die Komponente *AppComponent* aufgerufen werden.

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

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

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

Listing 3.1: Das Root-Module in der Datei app.module.ts

Feature Modulen ermöglichen die Gruppierung einer Anwendung in Anwendungsfällen. Mithilfe von Shared-Module können die Teile einer Anwendung zusammengefasst werden, die unabhängig vom Anwendungsfall verwendet werden können. (vgl. Freeman 2018, S. 528 ff.; vgl. *Introduction to modules*; vgl. Steyer et al. 2017, S. 105 ff.)

3.2.2. Komponenten

Komponenten sind Klassen, die Daten und Logik zur Anzeige in den zugehörigen Templates bereitstellen. Diese ermöglichen die Aufteilung einer Angular-Anwendung in logisch getrennte Teile. (vgl. Freeman 2018, S. 401)

Eine Komponente wird durch den Dekorator *@Component* gekennzeichnet und kann über verschiedene Dekorator-Eigenschaften konfiguriert werden. Die Eigenschaft *selector* identifiziert das HTML-Element, dass durch diese Komponente repräsentiert wird. Zur Anzeige der bereitgestellten Daten kann entweder ein Inline-Template *template* definiert oder auf ein externes Template *templateUrl* verwiesen werden. (vgl. *Introduction to components*; vgl. Freeman 2018, S. 405; vgl. Steyer et al. 2017, S. 47 ff.)

Beschreibung von Input- und Output Eigenschaften.

Die Beispielanwendung enthält die Komponenten *AppComponent* (siehe Listing 3.2) und *HelloComponent* (siehe Listing 3.3).

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

@Component({
selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent  {
  name: string;
}
```

Listing 3.2: Die Komponente AppComponent in der Datei app.component.ts

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

@Component({
  selector: 'hello',
  template: `<h1>Hello {{name}}!</h1>`,
  styles: [`h1 { font-family: Lato; }`]
})
export class HelloComponent  {
  @Input() name: string;
}
```

Listing 3.3: Die Komponente HelloComponent in der Datei hello.component.ts

3.2.3. Templates

Zur Darstellung von Komponenten nutzt Angular Templates. Ein Template besteht aus HTML Code, der um Angular spezifische Konzepte wie Direktiven, Datenbindungsausdrücke und Pipes erweitert wird. (vgl. *Introduction to components*; vgl. Steyer et al. 2017, S. 52)

Mit Direktiven kann einem Element zusätzliches Verhalten hinzugefügt werden. (vgl. Steyer et al. 2017, S. 265; vgl. Freeman 2018, S. 401) In Angular werden folgende drei Arten von Direktiven unterschieden. (vgl. *Attribute Directives*)

- Strukturelle Direktiven
- Attribut-Direktiven
- Komponenten

Die strukturellen Direktiven ändern die Struktur des zugehörigen HTML-Elements, indem sie HTML-Elemente hinzufügen oder entfernen. Hierfür verwenden die strukturellen Direktiven Templates, die beliebig oft gerendert werden. (vgl. Steyer et al. 2017, S. 269 ff.; vgl. Freeman 2018, S. 365) Beispiele für strukturellen Direktiven aus AngularJS sind (vgl. Freeman 2018, S. 261 ff.):

ngIf Fügt dem HTML-Dokument Inhalt hinzu, wenn die Bedingung wahr ist.

ngfor Fügt für jedes Item einer Datenquelle den gleichen Inhalt dem HTML-Dokument hinzu.

ngSwitch Fügt dem HTML-Dokument, abhängig vom Wert eines Ausdrucks, Inhalt hinzu.

Das Verhalten und das Aussehen des zugehörigen HTML-Elements kann durch die Attribut-Direktiven verändert werden. Diese Direktiven fügen oder entfernen dem zugehörigen HTML-Element Attribute. (vgl. ebd., S. 339) Beispiele für Attribut-Direktiven aus Angular-JS (vgl. ebd., S. 249 ff.):

ngStyle Mit dieser Direktive können unterschiedliche Style-Eigenschaften dem Element hinzugefügt werden.

ngClass Weißt dem Element ein oder mehrere Klassen hinzu.

Mittels einer Komponente kann einem HTML-Element eine View hinzugefügt werden. Komponenten sind nämlich Direktiven mit einer eigenen View. (vgl. Steyer et al. 2017, S. 265)

Die von Angular bereitgestellten Direktiven (engl. Built-In Directives) können durch selbst entwickelte Direktiven erweitert werden. (vgl. Freeman 2018, S. 261)

Der Datenaustausch zwischen der Komponente und dem Template erfolgt durch Datenbindungsausdrücke. Ein Datenbindungsausdruck bindet einen JavaScript-Ausdruck an ein Ziel. Das Ziel kann entweder eine Attribut-Direktive oder eine Eigenschaft des zugehörigen HTML-Elements sein. Der JavaScript-Ausdruck ermöglicht den Zugriff auf die Eigenschaften und Methoden der Komponente. (vgl. Freeman 2018, S. 237 ff.; vgl. Steyer et al. 2017, S. 52 f.; vgl. *Template Syntax*) Es gibt insgesamt drei Arten von Data-Bindings, die anhand der Flussrichtung der Daten unterschieden werden können.

Mit Pipes können die Daten vor der Ausgabe sortiert, formatiert oder gefiltert werden. (vgl. Steyer et al. 2017, S. 83 ff.)

Richtung	Syntax	Verwendung
One-way Komponente -> View	{{Ausdruck}} [Ziel]="Ausdruck"	Interpolation, Eigenschaft, Attribut, Klasse, Style
One-way Komponente <- View	(Ziel)="Ausdruck"	Events
Two-way	[(Ziel)]="Ausdruck"	Formular

Tabelle 3.1.: Arten von Data-Bindings

```
1 <hello name="{{name}}"></hello>
2 <label for="name">Name:</label>
3 <input name="name" [(ngModel)]="name">
```

Listing 3.4: Das Template in der Datei app.component.html

3.2.4. Services

Laut Freeman (vgl. 2018, S. 474) kann jedes Objekt, dass durch Dependency Injection verwaltet und verteilt wird, als Service bezeichnet werden. Services stellen wiederverwendbare Routinen oder Daten zur Verfügung, die von Direktiven, Komponenten, weiteren Services und Pipes verwendeten werden können. (vgl. Freeman 2018, S. 467 ff.; vgl. Steyer et al. 2017, S. 89)

Services verringern die Abhängigkeiten zwischen den Klassen, durch Dependency Injection. Hierdurch können Beispiel Unit-Tests einfacher durchgeführt werden. (vgl. Freeman 2018, S. 469)

Um einen Service zu verwenden, muss dieser entweder global in einem Modul oder lokal in einer Komponente registriert werden. Dies geschieht durch Einrichten eines Providers beim Modul oder der Komponente. Der Provider verknüpft ein Token mit einem Service. Ein Service kann eine Klasse, ein Wert, eine Funktion, eine Factory oder eine Weiterleitung sein. Aus diesem Grund gibt es unterschiedliche Provider.

Sobald ein Service global registrierte wurde, steht dieser auch in weiteren Modulen zur Verfügung. Ein lokal registrierter Service kann dahingegen nur von der jeweiligen Komponente und den direkten und indirekten Kindkomponenten verwendet werden.

Zur Nutzung eines Services muss die jeweilige Klasse den Service importieren und im Konstruktor deklarieren. Beim Erzeugen einer Instanz der Klasse injiziert AngularJS den jeweiligen Service.(vgl. Steyer et al. 2017, S. 92 ff.; vgl. Freeman 2018, S. 474 ff.)

4. ReactJS

4.1. Allgemein

ReactJS ist ein von Entwicklern des Unternehmens Facebook Inc. entwickeltes JavaScript Framework. **??**

4.2. Konzepte

4.2.1. Components

4.2.2. Lifecycle

4.2.3. Virtual DOM

4.3. Verwendung

5. OpenUI5

6. Fazit

A. Anhang

Literatur

Freeman, Adam (2018). *Pro Angular 6*. 3rd ed. Berkeley, CA: Apress. DOI: \url{10.1007/978-1-4842-3649-9}. URL: \url{http://dx.doi.org/10.1007/978-1-4842-3649-9}.

Google, Hrsg. *Attribute Directives*. URL: \url{https://angular.io/guide/attribute-directives}.

– Hrsg. *Introduction to components*. URL: \url{https://angular.io/guide/architecture-components}.

– Hrsg. *Introduction to modules*. URL: \url{https://angular.io/guide/architecture-modules}.

– Hrsg. *Template Syntax*. URL: \url{https://angular.io/guide/template-syntax}.

Steyer, Manfred und Daniel Schwab (2017). *Angular: Das Praxisbuch zu Grundlagen und Best Practices*. 2. Aufl. Heidelberg: O'Reilly.