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

3. Beispiel Webframeworks

3.1. AngularJS

3.1.1. Allgemein

Entwicklungsumgebung

3.1.2. Konzepte

Im Folgenden werden die in AngularJS verwendeten Konzepte näher erläutert. Die Beispielanwendung in Abbildung ?? 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:

```
├ example/
|    ├ src/
|        ├ app/
|        –app.component.css ⇒ CSS-Datei von AppComponent
|        –app.component.html ⇒ Template von AppComponent
|        –app.component.ts ⇒ Komponente AppComponent
|        –app.module.ts ⇒ Root-Modul AppModule
|        –hello.component.ts ⇒ Komponente HelloComponent
|        ├ assets/
|        ├ environments/
|    –index.html
```

Abbildung 1.: Screenshot Angular-Beispielanwendung

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

Module

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

Jede Angular-Anwendung besitzt ein Root-Modul, das die Anwendung konfiguriert. Wenn ein Browser die Beispielseite anfordert, dann schickt der Server den Inhalt der *index.html* als Antwort an den Browser zurück. Der Browser führt daraufhin die im HTML-Dokument enthaltenen Skript-Elemente aus. Dabei wird die Angular-Plattform initialisiert und das Root-Modul übergeben. (vgl. Steyer et al. 2017, S. 60; vgl. Freeman 2018, S. 226 ff.) In der Beispielanwendung stellt die Klasse *AppModule* siehe **??** das Root-Modul dar.

Module werden im Allgemeinen durch den Derokator *@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 ein weiteres Modul mit dem Namen *BrowserModule* und deklariert die zugehörige Komponente *AppComponent*. Diese Komponente soll auch beim Start der Anwendung aufgerufen werden.

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { FormsModule } from '@angular/forms';
4
```

```
5  import { AppComponent } from './app.component';
6  import { HelloComponent } from './hello.component';
7
8  @NgModule({
9  imports:      [ BrowserModule, FormsModule ],
10 declarations: [ AppComponent, HelloComponent ],
11 bootstrap:    [ AppComponent]
12 })
13 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.)

Komponenten und Templates

Komponenten sind Klassen, die Daten und Logik für die 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 (auch: Metadaten) 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.)

Für weitere Dekorator-Eigenschaften wird auf Freeman (2018, S. 405) verwiesen. Ein Beispiel für die Implementierung einer Komponente findet sich in Listing 3.2.

```
1  import { Component } from '@angular/core';
2
3  @Component({
4  selector: 'my-app',
5    templateUrl: './app.component.html',
6    styleUrls: [ './app.component.css' ]
7  })
8  export class AppComponent  {
9    name;
```

```
}
```

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

Zur Darstellung von Komponenten nutzt Angular Templates. Ein Template besteht aus HTML Code erweitert um Angular Ausdrücke. Das Template kann Pipes zur Formatierung von Daten, weitere Komponenten, Data-Binding Ausdrücke oder Direktiven enthalten. (vgl. *Introduction to components*; vgl. Steyer et al. 2017, S. 52)

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

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

Data-Binding Ausdrücke stellen eine Beziehung zwischen den Daten der Komponente und einem HTML-Element her. Hierdurch kann das Aussehen, der Inhalt oder das Verhalten dieses Elements dynamisch verändert werden. In Tabelle 3.1 werden die Arten von Data-Bindings unterschieden. Die Unterscheidung erfolgt mittels der Fließrichtung der Daten.

Das Ziel eines Data-Bindings ist entweder eine Attribut-Direktive oder eine Eigenschaft. Direktiven werden im nächsten Abschnitt näher erläutert. Der Ausdruck ist ein JavaScript-Fragement, der einen Zugriff auf die Eigenschaften und Methoden der Komponente ermöglicht. (vgl. Freeman 2018, S. 237 ff.; vgl. Steyer et al. 2017, S. 52 f.; vgl. *Template Syntax*)

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

Direktiven

Die im vorherigen Abschnitt beschriebenen Komponenten sind Direktiven mit einer eigenen View. Mit Direktiven kann einem Element zusätzliches Verhalten hinzugefügt werden. (vgl. Steyer et al. 2017, S. 265)[vgl.][401]Freeman.2018 In Angular werden folgende drei Arten von Direktiven unterschieden. (*Attribute Directives*, vgl.)

- Komponenten
- Attribut-Direktiven
- strukturelle Direktiven

Angular stellt Direktiven zur Verfügung (engl. Built-In Directives), die durch eigene Direktiven erweitert werden können. (vgl. Freeman 2018, S. 261)

Mit strukturellen Direktiven kann der Inhalt des HTML-Dokuments angepasst werden, indem Elemente dem diesem hinzugefügt oder entnommen werden. 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 (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.

Mit Attribut-Direktiven kann das Verhalten und Aussehen des zugehörigen Elementes angepasst werden, indem Attribute hinzugefügt oder entfernt werden. (vgl. Freeman 2018, 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.

Services

3.1.3. Verwendung

Einordnung in den Kontext

3.2. ReactJS

3.2.1. Allgemein

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

3.2.2. Konzepte

Components

Lifecycle

Virtual DOM

3.2.3. Verwendung

3.3. OpenUI5

4. Vergleich der Webframeworks

5. Bezug zum Projekt

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.