HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Fakultät Informatik und Medien

Studiengang Medieninformatik

Vergleichende Analyse der komponentenbasierten Frontend-Frameworks Angular und React

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

vorgelegt von

Felix Schmeißer

geb. am 22. 02. 1999

in München

69578

Verantwortlicher Hochschullehrer: Prof. Dr. rer. nat. Klaus Hering
Leipzig, Juni 2020 – September 2020

Erklärung

Ich versichere wahrheitsgemäß, diese Arbeit selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

.....
Felix Schmeißer
Leipzig, den 22. Juli 2020

Danksagung

Zunächst möchte ich meinem Betreuer Prof. Dr. rer. nat. Klaus Hering danken.

⋮

Zuletzt danke ich meinen Freunden, meinen Eltern sowie meiner Familie für die ständige Unterstützung während meines Studiums.

Inhaltsverzeichnis

Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Problemstellung

Es gibt unzählige JavaScript-Frameworks und es kommen ständig neue hinzu. Als Entwickler muss man abwägen, welche Lösung für das speziell vorliegende Szenario geeignet ist.

Angular ist ein umfangreiches Frontend-Framework und kann damit nahezu jede Aufgabe in diesem Bereich abdecken. Für sehr viele Problemstellungen gibt es eine Lösung direkt aus dem Framework. Die vorgesehene Archtitektur ist MVC bzw. MVVM und forciert damit eine strikte Trennung, die Entwicklung in großen Teams vereinfacht.

React als JavaScript-Bibilothek ist deutlich reduzierter. Abseits der elementaren Funktion von React werden Community-Erweiterungen verwendet. Demzufolge muss man hier zwischen verschiedenen Lösungsmöglichkeiten abwägen. React verwendet mit JSX eine JavaScript-Erweiterung, welche HTML und JavaScript kombiniert. Das macht die Entwicklung von Komponenten deutlich schneller, bricht allerdings mit MV*-Architekturen.

1.2 Ziel der Arbeit

Das Ziel der Arbeit ist, Gemeinsamkeiten und Unterschiede zwischen den Technologien herauszuarbeiten, auch mögliche Vor- und Nachteile hinsichtlich der Architektur werden angeschnitten. Dazu wird im Rahmen der Arbeit eine Testanwendung mit den Frameworks

implementiert. Durch Steuerung der Datenmenge und künstliche Geschwindigkeitsdrosselung können verschiedene Szenarien simuliert werden, um die Anwendungsbereiche der Frameworks einzugrenzen.

Welche Gemeinsamkeiten und Unterschiede besitzen Angular und React und welche Anwendungsbereiche ergeben sich aus der Performance in unterschiedlichen Auslastungsszenarien?

1.3 Vorgehensweise

Zunächst ein theoretischer Vergleich beider Technologien, der die grundlegenden Features beschreibt und gegenüberstellt. Im zweiten Schritt wird der Vergleich praktisch durchgeführt. Ziel ist es, eine identische Anwendung einmal in Angular und React zu implementieren, um die Unterschiede zu verdeutlichen und Grenzen aufzuzeigen. Im Anschluss werden verschiedene Testszenarien ausgewertet, um Anforderungen kleiner Anwendungen und größerer Enterprise-Produkte zu vergleichen. Abschließend werden die Beobachtungen eingeordnet und Schlüsse über Vor- und Nachteile beider Technologien gezogen. Zudem werden kurz Lösungsmöglichkeiten für etwaige Probleme diskutiert, damit einher geht ein Ausblick über die Erweiterbarkeit und Einbindung von externen Lösungen.

1.4 Inhaltlicher Aufbau der Arbeit

1.5 Abgrenzung

Diese Arbeit geht auf die essentiellen Funktionalitäten beider Frameworks ein und stellt die entsprechende Umsetzung im jeweils anderen vor. Fortgeschrittene Themen und

Erweiterungen werden nicht genauer betrachtet, werden aber mit Verweis auf offizielle Quellen in die Argumentation eingebunden.

Die Arbeit hat nicht den Anspruch, die teils kontroverse und nicht eindeutige Terminologie aufzuarbeiten. An dieser Stelle dient das Entwurfsmuster Model-View-Controller (MVC) als Beispiel. In der Literatur finden sich durchaus verschiedene Ansichten, ab wann eine gewählte Architektur dieses Entwurfsmuster erfüllt. Die Arbeit liefert für solche Begriffe eine begründete Einordnung, ohne den Anspruch zu haben, sämtliche Möglichkeiten abzubilden.

1.6 Begriffe

Frameworks

MVC

2 Allgemeines

2.1 Entstehung

2.1.1 Angular

Anfänge als Nebenprojekt

Die Historie von Angular beginnt 2009 als Nebenprojekt der Entwickler Miško Hevery und Adam Abrons. Ihn störte die aufwändige Entwicklung durch sehr viel Boilerplate-Code, u.a. Entwicklung einer Datenbank, Datenbankzugriff und Sicherheitsvorkehrungen. Die ursprüngliche Idee bestand also darin, diese Bestandteile zu abstrahieren und ein Framework zu entwickeln, welches von Designern ohne Programmierkenntnisse verwendet werden konnte. Anwendungen sollten unter dieser Prämisse mittels HTML umgesetzt werden können.

Quelle: Screenshot von: web.archive.org/web/20100227143939/

Abbildung 2.1: www.getangular.com

AngularJS

Hevery arbeitete zu dieser Zeit bei Google mit 2 weiteren Entwicklern an Google Feedback, die Umsetzung wurde mit einem eigenen Java Web Toolkit durchgeführt. In 6 Monaten entstanden 17.000 LOC, der immer schwerer zu warten und testen war. Daraufhin bat Hevery seinen Produktmanager um 2 Wochen Zeit, um das gesamte Projekt mit dem eigenen Framework neu zu entwickeln. Das gelang ihm in 3 Wochen, dabei konnte er den Code um 90% auf 1.500 LOC reduzieren. Google begann sich für Angular zu interessieren und stellte ein Entwicklerteam. Zunehmend wurden auch interne Projekte mit dem neu benannten AngularJS umgesetzt.[1] angularjs: https://angularjs.org/

Im Oktober 2010 wurde AngularJS mit der Versionsnummer 0.9.0 als erste stabile Version auf GitHub veröffentlicht. Bis zur Veröffentlichung der Version 1.0.0 im Juni 2012 war das Framework bereits auf breite Resonanz gestoßen. Das hat verschiedene Gründe, die sich im Ursprung auf die Überlegungen von Hevery und Abrons stützen. Am wichtigsten sind die folgenden Punkte, die AngularJS als erstes clientseitiges Webframework umsetzen konnte:

- Dependency Injection sorgt für lose Kopplung und vereinfacht damit Softwaretests
- Directives erlauben das Erstellen wiederverwendbarer HTML-Bausteine
- Zweiseitiges Data-Binding hält Model und View konsistent
- Das Nachladen im Browser wird hinfällig (Single-Page-Application)

https://www.ryadel.com/en/angular-angularjs-history-through-years-2009-2019/

Die Version 1.7.0 im Mai 2018 führte als letzter Release zu vorherigen Versionen inkompatible Änderungen ein. [1] Bis Juli 2021 wird AngularJS im Long Time Support unterstützt.

Es werden lediglich Sicherheitsfehler und durch neue Versionen der gängigen Browser erzwungene Bugs behoben.

https://de.wikipedia.org/wiki/AngularJS

Angular 2

2016 veröffentlichte Google den Nachfolger von AngularJS (Angular 2.0) und beschränkte die Bezeichnung auf Angular, da die neue Version als komplett neue Entwicklung nicht kompatibel zu vorhergehenden Releases war. Neu war vor allem der Wechsel zu TypeScript, einem Superset von JavaScript basierend auf ECMAScript6. Des Weiteren verschob sich der Fokus voll auf moderne Browser, um Workarounds zur Unterstützung veralteter Browser zu reduzieren. Viele Konzepte, mit denen AngularJS bereits Vorreiter im Bereich der Webframeworks war, wurden mit Angular weiter verbessert. Templates wurden weiter vereinfacht, u.A. wurden Bindings für Eigenschaften und Events deutlicher syntaktisch getrennt. Weiter wird Kapselung in Modulen und dynamisches Nachladen von Komponenten ermöglicht.

Der letzte Major Release (10.0.0) ist vom Juni 2020.

2.1.2 React

2.2 Verbreitung und Beliebtheit

2.3 Verwendete Sprachen

In den folgenden Abschnitten werden die von Angular und React angewendeten Webtechnologien eingeführt. Grundsätzlich ist das Resultat im Browser das Gleiche: ein

HTML-Dokument mit verknüpften CSS-Styles und JavaScript-Code für dynamische Funktionalitäten.

2.3.1 HTML-Templates

HTML gehört mit JavaScript und CSS zu den Grundsäulen von Webseiten und -anwendungen. Die **H**yper**t**ext **M**arkup **L**anguage gibt den Inhalten eine semantische Struktur und enthält Metainformationen. Grafische Darstellung und dynamische Änderungen werden mit CSS und JavaScript durchgeführt.

Angular Template Syntax

Angular führt eine spezielle Syntax ein, die den aktuellen HTML5-Standard um spezifische Funktionen erweitert. Das <script> *<script>*-Tag wird mit einer Warnung ignoriert, um Injection-Angriffe zu vermeiden. Sonst sind alle bekannten Tags des Standards erlaubt. Im folgenden Abschnitt werden einige wichtige Syntaxerweiterungen vorgestellt.

Interpolation und Ausdrücke

Mit Interpolation lassen sich Template Expressions (Ausdrücke) in der HTML-Template verwenden: <h1>Benutzername: userName </h1> Angular wertet die Ausdrücke, welche auch Rechenoperationen oder Funktionsaufrufe beinhalten können aus und ersetzt sie in der finalen Template durch einen String. Die Ausdrücke müssen sich also in Strings umwandeln lassen. Damit sind auch folgende Konstrukte erlaubt: 11 + getRandomNumber() = 11 + getRandomNumber(). Die Ausdrücke können neben Interpolationen auch an HTML-Properties gebunden werden: <img [src]=ïmageSourceUrl» Grundsätzlich ist beliebiger JavaScript-Code erlaubt. Nebeneffekte sollen vermieden werden, dadurch

fallen Zuweisungen (=, +=, -=, ...), einige Schlüsselwörter (a.A. new, typeof, instanceof), In- und Dekrementierung (++, –) und einige weitere Befehle ab dem ES2015-Standard weg. Auch Bitoperationen sind verboten, neu eingeführt werden der Pipe-Operator (|) zum Verwenden von Pipes und Null-Checks mit ! oder ?. Funktionen können theoretisch weiterhin Nebeneffekte verursachen. Neben der Inter Die Ausdrücke stehen dabei immer im Kontext der aktuellen Komponente, greifen also auf Properties der dazugehörigen TypeScript-Klasse zu. Das Data-Binding ist einseitig von Model zu View. Mehr dazu im folgenden Kapitel.

Das Gegenstück Template Expressions sind Template Statements (Anweisungen). Das Data-Binding ist ebenfalls einseitig von View zu Model. Anweisungen werden ausgeführt, wenn das korrespondierende Event aufgerufen wird, z.B. ein Klick-Event: <button (click)="login()»Einloggen</button> Anweisungen erlauben wie Ausdrücke beliebiges JavaScript, allerdings sind Nebeneffekte in diesem Fall der zentrale Punkt, um entsprechende Datenänderungen oder Navigation anzustoßen. Nicht erlaubt ist das Keyword new, In- und Dekrementierung, operative Zuweisungen (+=, -=), Bitoperationen und der Pipe-Operator. Der Kontext erstreckt sich ebenfalls auf die zugrunde liegende Komponente, erweitert um den Kontext der Template selbst. Anweisungen können folglich auch auf *$event*-Objekte und Template Variablen zugreifen. Mehr dazu gleich unter Built-in directives.

Banana in a Box Die letzte Möglichkeit zum Data-Binding ist zweiseitig. Wird also die View verändert, etwa durch Eingabeevents, dann wird das Model geupdated. Anders herum funktioniert das genauso. Ausdrücke werden dazu in zwei Klammern geschrieben: <input [(ngModel)]=üsername» Diese Schreibweise ist nur syntaktischer Zucker und ist mit folgender Schreibweise gleichzusetzen: <input [ngModel]=üsername"(ngModelChange)=üsername = $event» <input [value]=üsername"(input)=üsername = $event.target.value» Damit

werden die oben beschriebenen Bindings verwendet. https://angular.io/guide/template-syntax https://blog.thoughtram.io/angular/2016/10/13/two-way-data-binding-in-angular-2.html

JSX (React)

2.3.2 JavaScript

JavaScript (JS) ist die Programmiersprache des Internets. Alle gängigen Browser besitzen eine Engine zum Kompilieren von JavaScript-Code, Google Chrome setzt beispielsweise auf das eigene Open-Source-Projekt V8. JS wurde ursprünglich im Jahr 1995 von Brendan Eich entwickelt. 1997 wurde AngularJS als ECMAScript (ES) normiert. In den folgenden Jahren gab 2 weitere Versionen des Standards, 2009 kamen mit der 5. Version größere Änderungen. Die nächste Version war ES6 im Jahr 2015, ab hier gab es jährlich neue Releases.

Static Type Checking

TypeScript

TypeScript (TS) ebenfalls eine Implementierung des ECMAScript-Standards und wird von Microsoft entwickelt. Es erweitert JavaScript mit zusätzlichen Features, ist also eine Obermenge. JavaScript funktioniert auch in TypeScript, nicht umgekehrt. TS wird mit seinem Compiler in gültigen JavaScript-Quelltext in einer gewünschten Zielversion ab ES3 kompiliert.

Die neuen Features von TypeScript kennen Entwickler aus der objektorientierten Programmierung. TypeScript setzt, wie der Name schon andeutet, auf strengere Typisierung. Damit wird der Entwickler unterstützt und die Code-Qualität erhöht. Zwar gehen die Typings nach der Übersetzung verloren, allerdings werden Laufzeitfehler durch die vorgeschobene Kontrolle bereits bei der Entwicklung mit TypeScript reduziert.

https://blog.doubleslash.de/was-ist-eigentlich-typescript/

https://de.wikipedia.org/wiki/TypeScript

Angular wird komplett mit TypeScript geschrieben.

Flow

React legt sich nicht auf einen Type Checker fest. In der Dokumentation wird sowohl die Installation von TypeScript, als auch Flow beschrieben. Flow ist keine eigenständige Sprache wie TypeScript, sondern ermöglicht Type Checks durch Annotationen. Flow wird wie React von Facebook entwickelt. https://flow.org/en/docs/getting-started/ https://reactjs.org/docs/static-type-checking.html

2.3.3 CSS

2.4 Projekterstellung

2.4.1 Vorgehensweise

Angular CLI

Die Erstellung eines Angular Projektes erfolgt mit dem Angular CLI (Command Line Interface), welches mit dem Node Package Manager (NPM) installiert wird: *npm install -g @angular/cli*. Das CLI verfügt über einige Befehle, die beispielsweise externe Bibliotheken hinzufügen (*ng add <Bibliothek>*), neue Bestandteile (z.B. Components) anlegen (*ng generate <Schema>*) oder Tests starten (*ng test*). Mit *ng new <Projektname>* wird ein neues Projekt in einem entsprechend benannten Ordner angelegt. Das Projekt ist hier schon startbereit, mit *ng serve –open* kann man die Anwendung im Development Modus starten, durch das Flag wird automatisch der Browser geöffnet (http://localhost:4200/.). Änderungen im Projekt werden erkannt und die Anwendung sofort aktualisiert. https://angular.io/guide/setup-local

React Toolchains

React benötigt als grundsätzlich reine JavaScript-Bibliothek kein umfangreiches Setup. Die einfachste Möglichkeit der Einbindung besteht darin, React innerhalb des *<script>*-Tags einzubinden. Damit lässt sich React zu bestehenden Projekten hinzufügen, um etwa eine schrittweise Migration zu ermöglichen. Für neue Projekte die von Beginn an mit React entwickelt werden sollen, bietet sich eine Toolchain für das entsprechende Anwendungsszenario an. Damit wird die Umgebung für die React-Entwicklung optimiert:

Häufig wird bereits eine Grundstruktur und Scripte zum Starten der Anwendung mit erweiterten Debugoptionen generiert. Die einfachste Toolchain stammt ebenfalls von Facebook und heißt *create-react-app*. Mit dem Befehl *npx create-react-app <Projektname>* wie beim Angular CLI ein Projektordner generiert. Durch *npm start* wird die Applikation gehostet (http://localhost:3000/) https://github.com/facebook/create-react-app https://reactjs.org/docs/create-a-new-react-app.html#create-react-app

2.4.2 Aufbau

3 Technische Aspekte

Im folgenden Kapitel geht es um den konkretem technischen Vergleich zwischen Angular und React. Zunächst werden grundlegende Pattern und Modelle präsentiert, die in unterschiedlichem Maß und Form in die Entwicklung der Frameworks eingeflossen sind.

Die angeschnittenen Punkte decken nicht das gesamte Spektrum der verwendeten Technologien ab. Besonders React als erweiterbare JavaScript-Bibliothek forciert nur bedingt eine gewisse Architektur. Angular ist an diesem Punkt strikter, auf Entsprechungen für die Entwicklung mit React wird eingegangen.

3.1 Grundlegende Schnittstellen und Pattern

3.1.1 Components

2004 erscheint im Blog des Entwicklers und Architekturspezialisten Martin Fowler ein Artikel mit dem Titel "Inversion of Control Containers and the Dependency Injection pattern". In diesem Artikel beschreibt er u.A. das Entwurfsmuster Dependency Injection (DI) als präziseren Begriff für Inversion of Control. Die Erkenntnisse aus dieser Betrachtung definieren einige grundlegende Pattern von Angular, die in den folgenden Abschnitten erläutert werden. React ist nicht so stark an diese Konzepte gebunden, allerdings bauen gängige React-Bibliotheken auf diesen Mustern auf.

Components sind in der Terminologie nicht eineindeutig. Martin Fowler beschreibt seine Einordnung wie folgt: "I use component to mean a glob of software that's intended to be used,

without change, by an application that is out of the control of the writers of the component. By 'without change' I mean that the using application doesn't change the source code of the components, although they may alter the component's behavior by extending it in ways allowed by the component writers." (https://martinfowler.com/articles/injection.html#ComponentsAndSe
Mit diesem Ansatz kann man Komponenten in Angular ebenfalls betrachten. Eine Komponente ist in sich abgeschlossen und sollte von außen nicht verändert werden. Das Verhalten lässt sich allerdings ändern: Eine Beispielkomponente bietet eine Tabelle zum Anzeigen von Daten. Von außen lässt sich die Komponente bezüglich der Spalten, Zeilen, Styles und Inhalten in einem definierten Rahmen beeinflussen.

3.1.2 Dependency Injection

Das Entwurfsmuster "Dependency Injection"liefert einen Teil der Erklärung bereits mit seinem Namen: Abhängigkeiten werden ïnjiziert". Die Tabellenkomponente kann hier als Beispiel dienen:

```
class TableComponent {
    private finder: EntryFinder;

    constructor(){
        this.finder = new EntryFinderImpl();
    }

    public getFilteredData(filter: string): Entry[] {
        let allEntries: Entry[] = this.finder.getAllEntries();
        let filteredEntries: Entry[] = allEntries.filter(
```

```
            entry => entry.matches(filter)
            )

        return (filteredEntries);
    }
}

interface EntryFinder{
    findAll(): Entry[];
};
```

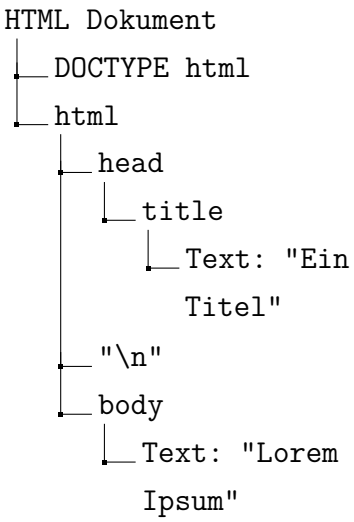
Das finder-Objekt liefert die benötigten Daten für unsere Tabelle. Damit muss sich die getFilteredData-Methode nicht die Beschaffung der Entry-Objekte kümmern, das liegt in der Implementation der Finder-Klasse. Mit einem Interface kann die Kapselung noch klarer gemacht werden, dennoch ist eine konkrete Implementation irgendwann notwendig (siehe Konstruktor). Abhängig von der konkreten Speicherung unserer Daten, z.B. als JSON oder SQL-Datenbank, ändert sich die konkrete Implementation. Die Komponente ist also idealerweise nur vom Interface abhängig.

3.1.3 Document Object Model

Das Document Object Model, kurz DOM, ist eine API, die den Zugriff und die Manipulation von HTML- und XML-Dokumenten erlaubt. Aus diesen Dokumenten lässt sich eine Baumstruktur ableiten:

Browser verarbeiten HTML-Dokumente zu diesen Bäumen, der DOM ist folglich der aktuelle Zustand der gezeigten Seite im Browser. Dynamische Änderungen der Seite, die

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Ein Titel
        </title>
    </head>
    <body>
        <span>
            Lorem Ipsum
        </span>
    </body>
</html>
```



mittels JavaScript umgesetzt werden, sind einfache Veränderungen des DOM über die Schnittstelle (`getElementById(...)` , `removeChild(...)`).

Virtual DOM

React verwendet einen Virtual DOM als Repräsentation des aktuellen Zustandes einer Komponente. Ändern sich dargestellte Werte oder Elemente, so muss der DOM neu aufgebaut und dargestellt (gerendert) werden. Gerade bei kleineren Änderungen ist es aber nicht nötig, den gesamten DOM neu zu rendern. Der virtuelle DOM wird mit dem echten DOM abgeglichen, daraufhin werden nur die veränderten Elemente und Kindelemente neu gerendert. https://reactjs.org/docs/faq-internals.html

3.2 Angular

3.2.1 Components and Directives

Pipes

3.2.2 Dependency Injection

Es gibt verschiedene Varianten der Dependency Injection, die von Angular verwendete Variante nennt sich Constructor Injection. Die Objekte, zu welchen eine Abhängigkeit der Komponente besteht, werden im Konstruktor übergeben. In der Regel nennt man diese Abhängigkeiten *Services*, es können allerdings auch Funktionen oder Werte injiziert werden.

```
constructor(private finderService: EntryFinderService){}
```

Angular initialisiert einen *Injector*, der einen Container mit Dependency-Instanzen verwaltet. Über einen Provider wird dem Injektor mitgeteilt, wie eine Instanz erzeugt wird, üblicherweise sind das die Klassen der Services. Wenn Angular eine neue Komponente erzeugt, dann wird überprüft, welche Abhängigkeiten bestehen. Gibt es Abhängigkeiten ohne bestehende Instanz, dann werden diese erzeugt und der Komponente zur Verfügung gestellt. Man erkennt, dass DI auf dem Singleton Pattern aufbaut. Damit werden Inkonsistenz und redundante Objekte vermieden.

https://angular.io/guide/architecture-services

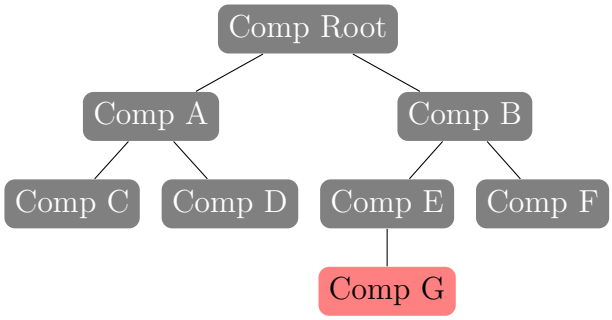


Abbildung 3.1: Komponentenbaum mit einem Veränderung in Komponente G

Services

3.2.3 Change Detection

https://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html
Change Detection versucht Veränderungen des Anwendungszustandes zu erkennen und die View entsprechend zu updaten. Dabei gibt es verschiedene Ansätze, die meisten Frontend Frameworks lösen das Problem auf verschiedenen Wegen. React nennt diesen Mechanismus Reconciliation, darum wird es in einem späteren Abschnitt gehen.

Change Detection funktioniert in Angular automatisch. Wenn man die Anwendung als Baum 3.1 mit Komponenten als Knoten betrachtet, dann führen verschiedene Auslöser zu einer Change Detection in diesem Baum.

```
abstract class ChangeDetectorRef {
    abstract markForCheck(): void
    abstract detach(): void
    abstract detectChanges(): void
    abstract checkNoChanges(): void
    abstract reattach(): void
```

```
}
```

https://www.mokkapps.de/blog/the-last-guide-for-angular-change-detection-you-will-ever-need/ https://indepth.dev/everything-you-need-to-know-about-change-detection-in-angular/

3.2.4 Lifecycle

3.2.5 State Management

3.2.6 Routing

3.2.7 NgModules

3.2.8 Projektarchitektur

https://medium.com/@cyrilletuzi/architecture-in-angular-projects-242606567e40

3.3 React

3.3.1 Reconciliation

Die Entwickler von React bezeichnen Reacts Change Detection mit Reconciliation. Eng verknüpft ist der Virtual DOM.

3.3.2 Lifecycle

3.3.3 State Management

3.3.4 Routing

3.3.5 Projektarchitektur

https://dev.to/jack/organizing-your-react-app-into-modules-d6n

4 Implementation

4.1 Anforderungsbeschreibung

4.1.1 Aufbau

4.1.2 Identische Teile

4.1.3 Umsetzungsdetails

Verwendete Features

Backend-Mockup

4.2 Angular

4.3 React

5 Performance Test

5.1 Testszenarien

5.2 Durchführung

6 Auswertung

6.1 Grundsätzlicher Vergleich

6.2 Probleme (Implementation)

6.3 Auswertung der Performance-Tests

6.4 Handlungsempfehlung

6.4.1 Anwendungsbereiche

6.4.2 Lernkurve

7 Fazit

7.1 Erfahrungen

7.2 Lernerfolge

7.3 Ausblick

Literatur

[1] ng-conf. *Miško Hevery and Brad Green - Keynote - NG-Conf 2014*. Jan. 2014. URL: https://www.youtube.com/watch?v=r1A1VR0ibIQ.

Anhang

Anhangsverzeichnis

Abbildungs- und Tabellenverzeichnis im Anhang

Abbildungen im Anhang

Tabellen im Anhang

A Anhangs

Abbildung A.1: Bildunterschrift im Anhang

Tabelle A.1: Tabellenüberschrift im Anhang