FH Vorarlberg
University of Applied Sciences

TBD

Bachelorarbeit II
zur Erlangung des akademischen Grades

Bachelor of Science in Engineering (BSc)

Fachhochschule Vorarlberg
Informatik – Software and Information Engineering

Betreut von
Prof. (FH) Dipl. Inform. Thomas Feilhauer

Vorgelegt von
Dominic Luidold
Dornbirn, 20. Mai 2021

Widmung

TODO

„TODO“
TODO

Kurzreferat

TODO

TODO

Abstract

TODO

TODO

Geschlechtergerechte Sprache

Der Verfasser der vorliegenden Arbeit bekennt sich zu einer geschlechtergerechten Sprachverwendung.

Um diese Arbeit sowohl geschlechtergerecht als auch -inklusive zu formulieren, werden explizit auf fix männlich oder weiblich zugeordnete Personengruppen, das sogennante Binnen-I oder andere Schreibweisen verzichtet. Stattdessen wird auf die Schreibweise mit einem Doppelpunkt (beispielsweise „Anwender:innen“, „Entwickler:innen“ etc.) gesetzt, die alle Personengruppen einschließt und dazu beiträgt, eine Bewusstheit für bestehende, diskriminierende Sprachgewohnheiten gegenüber Frauen sowie queeren Mitmenschen zu wecken beziehungsweise zu stärken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

DOM Document Object Model

JSON JavaScript Object Notation

MPA Multi-page Application

PWA Progressive Web App

SSR Server-side rendering

SPA Single-page Application

UI User Interface

UX User Experience

1 Einleitung

Diese Bachelorarbeit verfolgt das Ziel, einen Einblick in die Single-page Application (SPA) Frameworks *Angular*[1] und *Vaadin*[2] zu geben und deren Gemeinsamkeiten, Unterschiede sowie Vor- und Nachteile zu beleuchten.

Um ein grundlegendes Verständnis über die Thematik von Single-page Applications zu erlangen, wird zu Beginn der Arbeit auf das Konzept einer SPA eingegangen und die zugrundeliegende Herangehensweise mit der einer klassischen Multi-page Application (MPA) verglichen. Im weiteren Verlauf werden die unterschiedlichen Ansätze von Angular und Vaadin genauer betrachtet und eine tatsächliche Umsetzung der zuvor erläuterten Technologien mittels zweier Demo-Applikationen getestet. Am Ende dieser Arbeit wir darauf eingegangen, ob sich - anhand unterschiedlicher Kriterien und Anwendungsfälle - eine Empfehlung für eines der beiden SPA Frameworks aussprechen lässt.

1.1 Motivation

In den letzten Jahren lässt sich beobachten, dass Webapplikationen, Apps und Anwendungen allgemein verstärkt mittels des SPA-Ansatzes umgesetzt werden und somit auf einen Thin Client - im Gegensatz zu klassischeren Multi-page Applications - setzen. Für die Umsetzung einer solchen Applikation stehen eine Vielzahl von Frameworks zur Verfügung, die darüber hinaus weitere Features bieten und Entwickler:innen bei der Umsetzung unterstützen.

[1] Angular (https://angular.io)
[2] Vaadin (https://vaadin.com)

Die richtige Wahl des Frameworks, der jeweiligen Technologien und der im Hintergrund agierenden Strukturen spielen eine wesentliche Rolle bei der Planung und Umsetzung eines neuen Projektes. Welches Framework sich besser eignet, lässt sich oftmals nicht auf den ersten (oder sogar zweiten) Blick feststellen. Diese Arbeit befasst sich daher genauer mit dem Konzept von Single-page Applications und vergleicht zwei darauf aufbauende Frameworks, die mit deutlich unterschiedlichen Technologie-Stacks arbeiten und zu vergleichbaren Lösungen führen.

1.2 Problemstellung

Die in Abschnitt 1.1 auf Seite 10 angesprochene Vielzahl an SPA-Frameworks bietet grundlegend den Vorteil, dass eine große Auswahlmöglichkeit und eine gewisse Konkurrenz untereinander zu einem hohen Qualitätsstandard führt. Zudem wird dadurch sichergestellt, dass es für jedes Projekt - unabhängig von den jeweiligen Anforderungen und etwaigen Eigenheiten - eine Möglichkeit gibt, dieses mit einem der verfügbaren Frameworks umzusetzen. Auf der anderen Seite führt die stetig wachsende Anzahl an Möglichkeiten jedoch dazu, dass sich meist nur schwer beurteilen lässt, welches Framework sich für die Umsetzung einer Applikation bestmöglich eignet.

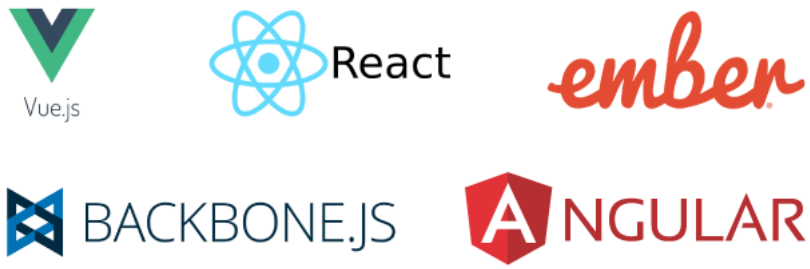

Abbildung 1.1: Liste möglicher JavaScript-Frameworks zur Umsetzung von Single-page Applications (Quelle: A. 2020)

Um eine geeignete Wahl eines Frameworks treffen zu können, sollten vorab Kriterien und Anforderungen definiert werden, die schlussendlich erfüllt werden

müssen. Neben grundlegenden Funktionalitäten, die in den meisten Fällen von einer Vielzahl der Frameworks abgedeckt werden können, stellen sich die projektspezifischen Eigenheiten und vor allem die Auswahl der zugrundeliegenden Technologien als eine der wichtigsten Herausforderungen dar. Diese Entscheidung muss gut überlegt und abgewogen werden, da diese im weiteren Verlauf weitreichende Folgen bei der Umsetzung einer (Web-) Applikation zur Folge hat und sich ein Wechsel nach gestarteter Entwicklung nur unter großem Aufwand umsetzen lässt.

Die in Abbildung 1.1 auf Seite 11 dargestellten Frameworks zeigen eine Auswahl an Frameworks auf, die auf *JavaScript* aufbauen beziehungsweise basieren und somit primär auf dem Client - dem Browser - eingesetzt werden können. Single-page Applications lassen sich jedoch nicht nur Frontend-seitig entwickeln (bei denen ein Großteil der Logik auf einem externen Server abläuft), sondern können ebenfalls mittels auf *Java* basierenden Frameworks umgesetzt werden. Bei diesen Frameworks - zu denen unter anderem Vaadin gehört - lässt sich sowohl die Logik als auch das User Interface (UI), stellenweise gänzlich, kombinieren.

Da die unterschiedlichen Ansätze, sowohl hinter Vaadin als auch Angular, gewisse Vor- und Nachteile sowie Tücken mit sich bringen, fällt die Wahl auf eines der beiden SPA-fähigen Frameworks auf den ersten Blick nicht leicht. Hinzu kommt die Frage, welches der Frameworks weiterführende Funktionalitäten bietet, um mit geringem Aufwand beispielsweise eine Progressive Web App (PWA) umzusetzen oder anwendungsspezifische Daten lokal sowie extern persistieren zu können.

1.3 Zielsetzung

Die in den Abschnitten 1.1 und 1.2 angeführten Punkte haben aufgezeigt, dass die große Anzahl an Frameworks, mit denen Single-page Applications umgesetzt werden können, zwar sehr positiv einzuschätzen ist, die damit verbundenen Probleme bei der Auswahl des richtigen Frameworks werden dadurch jedoch verstärkt. Aufgrund der unterschiedlichen zugrundeliegenden Technologien und einhergehenden Herangehensweisen ist eine bedachte Wahl wichtig.

Diese Arbeit verfolgt daher das Ziel, das JavaScript Framework *Angular* dem auf Java basierenden Framework *Vaadin* gegenüberzustellen und zu vergleichen. Das Ziel ist es, mittels Literatur belegter Vergleiche einen Allgemeinen Überblick über Single-page Applications zu geben, diese klassischen Ansätzen gegenüberzustellen und zwei Demo-Applikationen zu entwickeln. Diese Webanwendungen werden dann herangezogen, um anhand von vorab definierten Kriterien feststellen zu können, ob und in wie weit Empfehlungen für eines der beiden Frameworks ausgesprochen werden kann.

Um den Fokus dieser Arbeit genauer zu definieren und einzuschränken, wird die Planung, Umsetzung sowie abschließenden Beurteilung der Applikationen anhand der ausgearbeiteten Kriterien auf folgende Punkte beschränkt:

- Möglichkeit zur einfachen Umsetzung einer Progressive Web App (PWA)
- Möglichkeit der Wiederverwendbarkeit von Komponenten, gegebenenfalls mittels *Web Components*
- Möglichkeit Daten lokal (Browser) sowie extern (Server) zu persistieren

2 Stand der Technik

Das folgende Kapitel gibt einen Überblick über die Funktionsweise einer Single-page Application und vergleicht das Konzept von SPAs mit klassischen Multi-page Applications. Im Anschluss wird im Detail auf die Funktionsweise von Angular und Vaadin beziehungsweise deren unterschiedlichen Ansätze in Hinblick auf Entwicklung der UI mittels JavaScript und Java eingegangen. Im weiteren Verlauf werden die damit verbundenen Vor- und Nachteile genauer beleuchtet.

2.1 Konzept einer Single-page Application

Eine klassische Multi-page Application basiert auf dem Konzept, dass bei jedem Aufruf eines neuen View beziehungsweise einer HTML-Seite eine Anfrage an den Server gestellt wird. Dieser verarbeitet die Anfrage und retourniert das jeweils neu zusammengestellte Resultat der Präsentations- sowie der darunterliegenden Schichten an den Client. Eine Single-page Application ist hingegen eine Webanwendung, bei der die Präsentationsschicht und die damit verbundene Logik vom Server entkoppelt und vollständig in den Client, sprich den Browser, ausgelagert wird. (Scott 2015, Seite 5ff.)

Die Abbildung 2.1 auf Seite 15 stellt den Aufbau solch einer SPA schematisch dar und verdeutlicht, dass die Darstellung der mittels AJAX und XHR angeforderten Daten komplett vom Client übernommen wird. Serverseitig wird lediglich ein *Controller* benötigt, welcher die übermittelten Daten in für die Logik entsprechend verständliche Objekte umwandeln kann.

Diese Herangehensweise führt dazu, dass beim Aufrufen einer (Unter-)Seite keine komplett neue HTML-Seite geladen werden muss, sondern lediglich Teile des User Interface - sogennante *Views* - ausgetauscht werden. Hierfür wird das

entsprechende Document Object Model (DOM) mittels JavaScript dynamisch ausgetauscht und die benötigten Daten werden bei Bedarf asynchron mittels AJAX und XHR vom Server geladen. (Scott 2015, Seite 7)

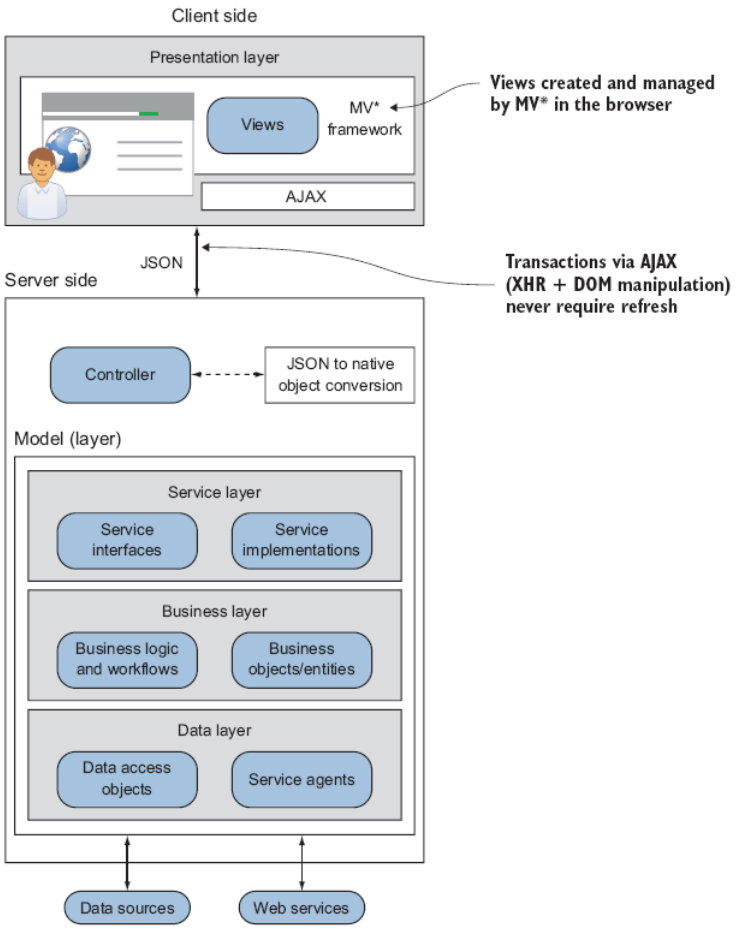

Abbildung 2.1: Aufbau einer Single-page Application
(Quelle: Scott 2015, Seite 6)

2.1.1 Server-side rendering

Neben der reinen Übertragung von Daten mittels JSON (oder anderweitigen Datenformaten) kann bei SPAs alternativ beziehungsweise erweiternd auch auf **Server-side rendering (SSR)** gesetzt werden. Bei diesem Ansatz werden Ausschnitte von HTML bereits auf dem Server vorbereitet und zusammen mit

weiterführenden Daten an den Client geschickt. Dieser kann somit einen Teil der Antwort ohne weitere Aufbereitung darstellen, während die restlichen Daten mittels DOM-Manipulation in die View eingebettet werden. (Scott 2015, Seite 7)

2.1.2 Bestandteile einer SPA

Der zugrundeliegende Aufbau einer Single-page Application - und der Bestandteil der Applikation, der lediglich einmal geladen wird - ist die sogenannte *Shell*. Die Shell ist eine einzelne HTML-Datei, welche vom Browser vollständig geladen wird und in den meisten Fällen lediglich minimale Strukturen sowie einen leeren `DIV` Tag enthält, wie Abbildung 2.2 auf Seite 15 zeigt. Genutzt wird diese als Ausgangspunkt für alle weiteren Views, die unabhängig von der Shell agieren und dynamisch geladen werden. (Scott 2015, Seite 8)

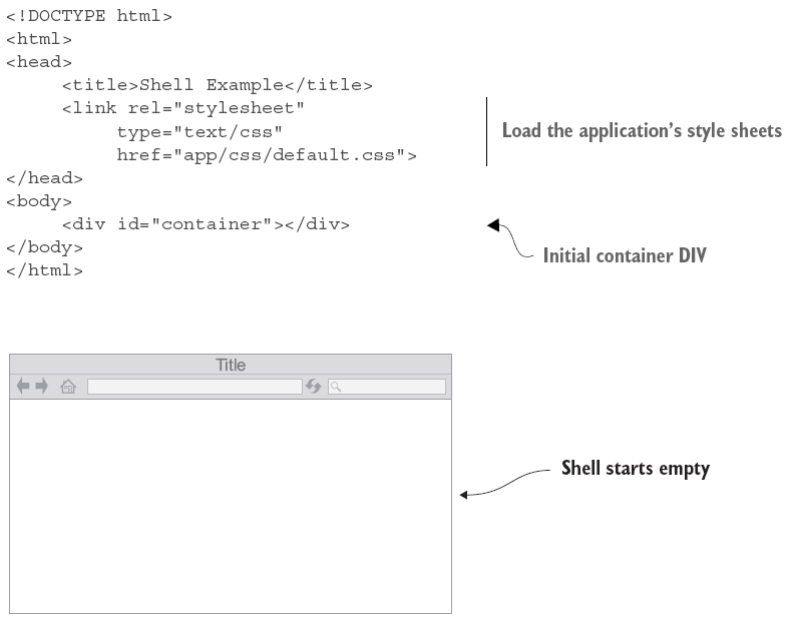

Abbildung 2.2: Single-page Application Shell
(Quelle: Scott 2015, Seite 8)

Voneinander getrennt sichtbare Bereiche der Anwendung können im weiteren Verlauf ebenfalls mit `DIV` Tags abgegrenzt werden und sind dem in der Shell definierten `DIV` Container untergeordnet. Dies ermöglicht sowohl eine logische

als auch inhaltliche Gruppierung und das gezielte Austauschen bestimmter Bereiche, sogenannter *Regions*. (Scott 2015, Seite 9)

Die einzeln dargestellten Views, welche dynamisch ausgetauscht werden können, stellen keine vollständigen HTML-Seiten dar, sondern bilden lediglich gezielt definierte Ausschnitte. Diese Ausschnitte werden bei jedem Navigationsvorgang innerhalb der Website durch das entsprechend eingesetzte Framework ausgetauscht und erfordern kein erneutes Laden der Website. (Scott 2015, Seite 10f.)

2.2 Vorteile einer SPA gegenüber einer MPA

Der Einsatz und die Entwicklung einer Single-page Application bietet sowohl Vorteile für Entwickler:innen als auch Anwender:innen gegenüber der Nutzung einer klassischen Multi-page Application.

Der bereits mehrfach angesprochene Vorgang, lediglich bestimmte Teile beziehungsweise Views der Webanwendung auszutauschen, erhöht die Benutzbarkeit sowie die User Experience (UX) laut Mikowski und Powell deutlich. Da keine komplett neue (Unter-)Seite geladen werden muss, entfällt das Aufscheinen einer - abhängig von der Internet- und Servergeschwindigkeit - kurz sichtbaren, weißen Übergangsseite während des Ladeprozesses. Dem:Der Benutzer:in kann stattdessen beispielsweise ein dynamisch dargestellter Fortschrittsbalken dargestellt werden, der sich bei vorhandener Ladezeit laufend aktualisiert. (Mikowski und Powell 2013, Seite 20)

Scott hebt hervor, dass die Aufteilung in eine entkoppelte Präsentationsschicht auf dem Client dazu führt, dass diese unabhängig von der Logik auf dem Server gewartet und aktualisiert werden kann. Während bei klassischen MPAs stellenweise HTML, JavaScript ect. mit serverseitigem Code (beispielsweise *PHP*, *JavaServer Pages*, ...) vermischt werden, kann bei SPAs zudem eine gewisse Differenzierung und Abtrennung von HTML, CSS und JavaScript im Frontend erzielt werden, was die Wartbarkeit ebenfalls erhöht. (Scott 2015, Seite 13)

Sowohl Mikowski und Powell als auch Scott gehen des Weiteren darauf ein, dass die Datenübertragung und Verarbeitung bei einer Single-page Application effizienter und schneller stattfinden kann, als bei einer Multi-page Application. Die Programmlogik zur Darstellung und dynamischen Entscheidungsfindung befindet sich beim Client (weshalb dieser Operationen schnell durchführen kann), während der Server lediglich Validierung, Authentifizierung und Datenspeicherung durchführt. (Mikowski und Powell 2013, Seite 20) Zudem sind Transaktionen zwischen Client und Server nach dem Initialen Aufruf der Applikation schneller, da lediglich Daten in einem vorab definierten Datenformat asynchron übertragen und keine kompletten HTML-Seiten samt JavaScript und CSS ausgetauscht werden müssen. (Mikowski und Powell 2013, Seite 21)

2.3 Funktionsweise von Vaadin

Sowohl Angular als auch Vaadin sind beides Frameworks, die das Entwickeln von Single-page Applications unterstützen und ermöglichen. Der gewählte Ansatz, die zugrundeliegenden Technologien und die jeweiligen Herangehensweisen unterscheiden sich stellenweise jedoch deutlich voneinander.

Der nachfolgende Abschnitt befasst sich daher im Detail mit der Funktionsweise und den Konzepten von Vaadin sowohl aus dem Blickwinkel von Entwickler:innen als auch Anwender:innen. Wo sinnvoll, wird ein direkter Vergleich zu Angular gezogen und darauf eingegangen, wie die beiden Frameworks Probleme unterschiedlich handhaben und lösen.

2.3.1 Ansätze von Vaadin

Vaadin ist ein Framework beziehungsweise eine Plattform, mit der Webapplikationen sowohl komplett in Java, vollständig mit TypeScript und HTML als auch in einer Kombination beider entwickelt und umgesetzt werden können. Die beiden Ansätze sind dabei in zwei verschiedene Frameworks aufgeteilt, um - je nach Einsatzzweck - diese entsprechend einsetzen zu können:

- **Vaadin Flow** ist ein Framework, mit dem Webapplikationen in Java umgesetzt werden können. Technologien wie beispielsweise HTML oder

JavaScript werden bei der Entwicklungen der UI nicht benötigt, der gesamte Programmcode basiert auf einer einheitlichen Programmiersprache. Die gesamte Anwendung selbst läuft auf einem Server, während das Framework selbst den Applikationszustand sowie die Client-Server-Kommunikation übernimmt. (Vaadin Ltd 2021b)

- **Vaadin Fusion** ist ein Framework, bei dem TypeScript für das Frontend auf dem Client und Java für das Backend auf dem Server eingesetzt wird. Mit Fusion können clientseitig reaktive Webapplikationen entwickelt werden, die typsichere Java Endpunkte aufrufen. Vorgefertigte Komponenten erleichtern hierbei das Entwickeln der UI, während eigene Elemente mittels voller Kontrolle über das DOM umgesetzt werden können. (Vaadin Ltd 2021c)

Diese Arbeit befasst sich nachfolgend primär mit *Vaadin Flow*, um eine alternative Herangehensweise aufzuzeigen, wie eine Single-page Application komplett auf dem Server und mittels Java entwickelt werden kann. Durch diesen Fokus kann im weiteren Verlauf ein tiefergehenderer Vergleich den Funktionalitäten und Konzepten von Vaadin und Angular bewerkstelligt werden, der bei spezifischen Punkten konkret auf die Unterschiede der beiden Technologien eingeht.

2.3.2 Frontend und Backend in Java

Vaadin selbst schreibt, dass sich das Arbeiten mit HTML, CSS und JavaScript für reine Java-Entwickler sowohl als herausfordernd als auch als zeitintensiv darstellt. (Vaadin Ltd 2021a, Framework - Introduction - Overview)

Vaadin Flow bietet daher die Möglichkeit, mit einer vollständig in Java geschriebenen Applikation - die auf einem Server ausfgeführt werden kann - jeweils die Anwendungslogik sowie das User Interface umzusetzen. Die Vielzahl von sogenannten *Components*, welche Vaadin von Haus aus mitliefert und einzelnen Bestandteilen wie beispielsweise einem Button oder ähnlichem entspricht, unterstützen Entwickler:innen dabei, bei Bedarf ohne HTML oder JavaScript auszukommen. Die Components steuern dabei das zugrundeliegende JavaScript

im Hintergrund über die Framework-eigene *Java API* beziehungsweise die *Java Component API*. Der Quellcode 1 auf Seite 20 stellt einen stark vereinfachten Ausschnitt von Components dar, die so im Client bereits entsprechend dargestellt werden können. (Vaadin Ltd 2021a, Framework - Introduction - Overview)

```
1  // Create an HTML element
2  Div layout = new Div();
3
4  // Use TextField for standard text input
5  TextField textField = new TextField("Your name");
6
7  // Button click listeners can be defined as lambda expressions
8  Button button = new Button("Say hello",
9            e -> Notification.show("Hello!"));
10
11 // Add the web components to the HTML element
12 layout.add(textField, button);
```

Quellcode 1: Beispiel einer einfachen UI mittels der *Java API* (Quelle: Vaadin Ltd 2021a, Framework - Introduction - Overview)

Während mit Vaadin Flow hauptsächlich in Java, und somit auf dem Server, entwickelt wird, bietet das Framework dennoch die Möglichkeit, auf Browser APIs, spezifische Web Components sowie das DOM zuzugreifen.

Als Vorteil stellt sich zudem heraus, dass die Anbindung des Frontends an ein Backend in den meisten Fällen mit Vaadin Flow bereits komplett vorhanden ist und nicht separat mit auf REST basierender Kommunikation umgesetzt werden muss. Dadurch kann die UI und die dafür benötigten Daten direkt über Java verknüpft und auch aktualisiert werde. Hierbei unterstützt das mitgelieferte *Two-way data binding* die Entwicklung und Benutzbarkeit, indem Änderungen auf dem Client automatisch auch auf dem Server - und umgekehrt - abgebildet werden. (Vaadin Ltd 2021a, Framework - Introduction - Overview)

2.3.3 Kommunikation zwischen Client und Server

Bei einer Single-page Application spielt die Kommunikation des Clients (dem JavaScript/TypeScript Frontend) und dem Server (dem Backend) eine sehr

wichtige Rolle. Im Gegensatz zu Multi-page Applications, bei denen die Daten bereits auf dem Server verarbeitet, eingebettet und als Ganzes übertragen werden, werden bei SPAs diese erst bei Bedarf und im Nachhinein geladen. Die Art und Weise, wie dieser Vorgang umgesetzt wird, unterscheidet sich bei Vaadin Flow und Angular jedoch deutlich voneinander.

2.3.3.1 Herangehensweise von Vaadin

Da sowohl die Persitenzschicht, die Applikationslogik als auch das User Interface bei Vaadin Flow mittels Java umgesetzt werden können, ergibt sich der Vorteil, dass die Kommunikation zwischen Client und Server vom Framework selbst übernommen wird und hierbei auf das bereits angesprochene *Two-way data binding* setzt. Die Nutzung von Vaadin-eigenen Components bietet des Weiteren die Möglichkeit, das Document Object Model im Webbrowser selbst zu steuern, während eine Repräsentation desselben DOM serverseitig in Java gehalten wird. Änderungen, die auf dem Client durchgeführt werden, werden automatisch synchronisiert. (Vaadin Ltd 2021a, Framework - Introduction - Core Concepts)

Artur Signell, CTO der Vaadin Ltd., erklärt in einem Foren-Beitrag vom Juni 2018, dass genauere Informationen zur tatsächlich Umsetzung der automatisch vorgenommen Kommunikation nicht vorhanden ist, da die Umsetzung davon als rein internes Implementierungsdetail gehandhabt wird. (Signell 2018)

Mittels den Entwicklertools, die in gängigen Webbrowsern zur Verfügung stehen, kann sich - abseits der nicht vorhandenen Dokumentation - jedoch ein kurzes Bild davon gemacht werden, wie die Kommunikation grundsätzlich funktioniert. Die Abbildung 2.3 auf Seite 22 zeigt die vom Client an den Server geschickte Anfrage, die durch das Befüllen eines Vaadin `TextField` (einem HTML - `INPUT` Element) ausgelöst wird. Für die Kommunikation wird JSON eingesetzt, welches die in das `INPUT` Element eingefüllten Daten an den Server schickt, auf die schlussendlich mittels Java-typischer Notation zugegriffen werden kann.

Abbildung 2.3: Automatisch erzeugte Kommunikation von Vaadin Flow (Quelle: eigene Abbildung)

2.3.3.2 Herangehensweise von Angular

Im Vergleich zur automatischen Kommunikation, die Vaadin Flow von Haus aus bietet, unterstützt Angular Entwickler:innen zwar beim Datenaustausch mit einem Server, die konkrete Umsetzung muss jedoch deutlich eigenständiger programmiert werden.

Der von Angular entwickelte `HttpClient` Service - der über das `@angular/common/http` Package bezogen werden kann - stellt entsprechend eine API zur Verfügung, mit der typisierte `Response` Objekte angefordert werden können, eine vereinheitlichte Fehlerbehandlung ermöglicht wird sowie das Abfangen und bearbeiten von `Requests` und `Responses` erlaubt. (Google LLC 2021)

Der Einsatz der angesprochenen API kann bei Angular grundsätzlich im gesamten Projekt erfolgen, Wilken empfiehlt jedoch, die Nutzung von der tatsächlichen Logik zu abstrahieren und dafür einen eigenständigen Service zu erstellen. Der `HttpClient`, welcher zur Kommunikation mit einem Server verwendet wird, unterstützt dabei HTTP Request-Methoden wie `GET`, `POST`, `PUT` und `DELETE`, die bei einem entsprechenden Aufruf ein `Observable` als Antwort liefern. Mit diesem kann in weiterer Folge in der Applikation gearbeitet und

gegebenenfalls auf die Daten zugegriffen werden. (Wilken 2018, Seite 142-144.)

Der Quellcode 2 auf Seite 23 zeigt einen exemplarischen Service, der mittels `HTTP GET` eine Anfrage an den Server beziehungsweise einen API Endpunkt stellt und ein Array von `Sample` als Antwort zurückliefert.

```
[...]

@Injectable()
export class SampleService {
    constructor(private http: HttpClient) {}

    getSampleData() {
        return this.http.get<Array<Sample>>(/* URL to server or API endpoint */);
    }
}

[...]
```

Quellcode 2: Exemplarische Nutzung des `HttpClient` in einem Service

Der Quellcode 3 auf Seite 23 zeigt in Folge die Nutzung der gerade eben demonstrierten Funktion, bei der die Daten mittels `subscribe()` abgefragt werden können und entsprechend zugewiesen werden, sobald diese zur Verfügung stehen.

```
[...]

private loadSampleData() {
    this.sampleService.getSampleData().subscribe(data => {this.data = data});
}

[...]
```

Quellcode 3: Beispielhafte Nutzung der `getSampleData` Funktion

Verglichen mit der im Abschnitt 2.3.3.1 auf Seite 21 beschriebenen Herangehensweise von Vaadin unterscheidet sich Angular deutlich. Wilken merkt an, dass die Nutzung des `HttpClient` der am häufigsten verwendete Ansatz bei Angular darstellt. Um eine Kommunikation mit einem Server herzustellen, können laut ihm jedoch auch diverse andere Protokolle und Technologien genutzt werden. (Wilken 2018)

Literatur

A., Sviatoslav (Jan. 2020). *The Best JS Frameworks for Front End*. URL: `https://rubygarage.org/blog/best-javascript-frameworks-for-front-end` (besucht am 08. 02. 2021).

Google LLC (2021). *Communicating with backend services using HTTP*. Englisch. URL: `https://angular.io/guide/http` (besucht am 24. 02. 2021).

Mikowski, Michael und Josh Powell (Sep. 2013). *Single Page Web Applications: JavaScript end-to-end*. Englisch. 1st Edition. Shelter Island, NY: Manning Publications. ISBN: 978-1-61729-075-6.

Scott, Emmit (Nov. 2015). *SPA Design and Architecture: Understanding Single Page Web Applications*. Englisch. 1st Edition. Shelter Island, NY: Manning Publications. ISBN: 978-1-61729-243-9.

Signell, Artur (Juni 2018). *Explanation of Flow's 2-way communication protocol? | 1. Vaadin Flow - Java | Forum*. Englisch. URL: `https://vaadin.com/forum/thread/17118150/17119448` (besucht am 23. 02. 2021).

Vaadin Ltd (2021a). *Documentation | Vaadin 18 Docs*. Englisch. URL: `https://vaadin.com/docs/v18/` (besucht am 22. 02. 2021).

– (2021b). *Vaadin Flow - Modern web apps in Java*. Englisch. URL: `https://vaadin.com/flow` (besucht am 22. 02. 2021).

– (2021c). *Vaadin Fusion*. Englisch. URL: `https://vaadin.com/fusion` (besucht am 22. 02. 2021).

Wilken, Jeremy (Apr. 2018). *Angular in Action*. Englisch. 1st Edition. Shelter Island, NY: Manning Publications. ISBN: 978-1-61729-331-3.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit I selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am 20. Mai 2021 Dominic Luidold