Hauptseminar-Arbeit

Vue.js

Prüfungsleitung des Moduls
CS1025 Hauptseminar

von

Maximilian Biebl
`Matrikelnr.: 5323481`

am 23. Mai 2023

Dozent: Sebastian Süß

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Limburg, 23. Mai 2023

Zusammenfassung

Eine Webanwendung mit einer komplexen Benutzeroberfläche, ohne die Verwendung eines Frontend-Frameworks zu entwickeln, ist aufwendig und fehleranfällig. Zudem besteht ein erhöhtes Risiko von Inkonsistenzen in Bezug auf Inhalt und Qualität, was die spätere Wartung erschwert. Für eine Webanwendung wird in der Regel ein Frontend-Framework verwendet, die aktuell verbreitetsten dieser Frameworks sind React.js, Angular und Vue.js. Im Jahr 2013 startete der Chinese Evan You die Entwicklung von Vue.js, mit der Absicht ein kleineres, flexibleres Angular zu entwickeln. In dieser Arbeit wird sich genauer mit Vue.js auseinandergesetzt und ein Vergleich anhand einer in Angular und Vue.js implementierten Einkaufslistenapp durchgeführt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Frontend Frameworks

In diesem Kapitel wird zunächst allgemein in die JavaScript-Frontend-Frameworks eingeführt. Die Drei aktuell gängigsten Frontend-Frameworks React.js, Angular und Vue.js werden vorgestellt.

An eine moderne Webapplikation werden hohe Anforderungen an den Funktionsumfang durch den Endnutzer gestellt. Die Entwickler einer solchen Webapplikation erwarten eine einheitliche Codequalität und einheitliche Strukturen für eine bessere Wartbarkeit der Webapplikation. Um beim Erstellen neuer Webapplikationen die Entwickler dabei zu unterstützen, diese Ziele zu erfüllen, werden unter anderem entsprechende Frontend Frameworks verwendet.

> Front-end frameworks determine the logic, structure, design, behavior, and animation of every element you see on-screen when you interact with websites, web applications, and mobile apps. [1]

Man kann zwischen UI-Frameworks in HTML und CSS sowie JavaScript-Frameworks unterscheiden. Letztere dienen als Verbindung zwischen der Darstellung und der Logik im Frontend.
JavaScript-Frameworks beeinflussen den HTML DOM und können HTML sowie CSS Elemente verändern und mit diesen interagieren. Beim HTML DOM (Document Object Model) handelt es um die hierarchische Anordnung einer HTML-Seite. [14, S. 26-27]
Aus den Statistiken 1.2 und 1.1 lässt sich schließen, dass die drei aktuell gängigsten JavaScript-Frontend-Frameworks React.js, Angular und Vue.js sind.

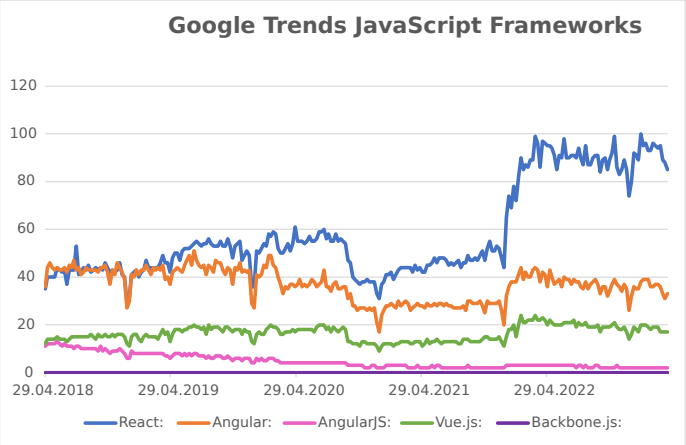

Abbildung 1.1: Google Trends zur Häufigkeit von Suchanfragen nach Frameworks der letzten 5 Jahre [2]

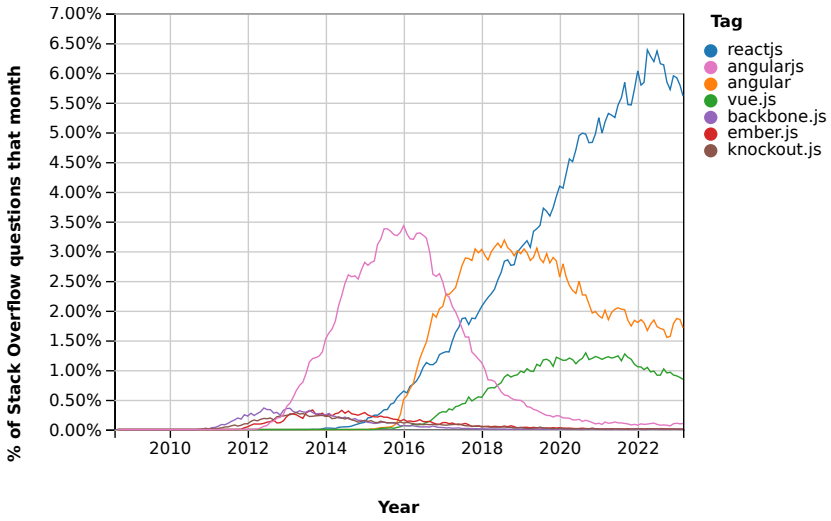

Abbildung 1.2: Stack Overflow Statistik zur Häufigkeit von Fragen nach Frameworks [3]

1.1 Vue.js

Vue.js wurde von Evan You zunächst als Nebenprojekt entwickelt und erschien 2014. Das Projekt finanziert sich laut eigenen Angaben aus Spenden und wird von sowohl Vollzeitentwicklern und Freiwilligen gepflegt. [7]
Nach Google Trends [2] und Stack Overflow Statistik [3] ist es aktuell das drittbeliebteste Frontend Framework. Hohe Beliebtheit hat das Framework in China (siehe Abb.1.3). Besonderheit des Frameworks ist, dass es in seiner Grundform keine 20 KB groß ist [6, S. 523].

1.2 Angular

Angular wurde von Google entwickelt und erschien 2016 ursprünglich als *AngularJs 2.0*. Bei Angular handelt sich um eine vollständige Neuimplementierung in TypeScript, unabhängig der Codebasis des Vorgängers AngularJS. Die Pflege und Weiterentwicklung wird vom Angular Team von Google durchgeführt. [6, S. 209-210]
Nach Google Trends [2] und Stack Overflow Statistik [3] ist es aktuell das zweitbeliebteste Frontend Framework.

1.3 React.js

Das laut Stack Overflow Statistik [3] und Google Trends [2] gefragteste Frontend Framework ist aktuell React.js. React.js wurde von Facebook entwickelt und 2013 veröffentlicht. Das Ziel bestand darin, ein Framework zu entwickeln, das den Anforderungen an Skalierbarkeit und Wartbarkeit gerecht wird, wie sie für eine umfangreiche Webapplikation wie Facebook erforderlich ist [4, S. 1]. Mittlerweile ist es ein Open-Source-Projekt des Facebookmutterkonzerns Meta. React.js setzt dabei auf *JavaScript syntax extension* (JSX), was einen hybriden Code aus JavaScript und HTML-Elementen ermöglicht. [5]

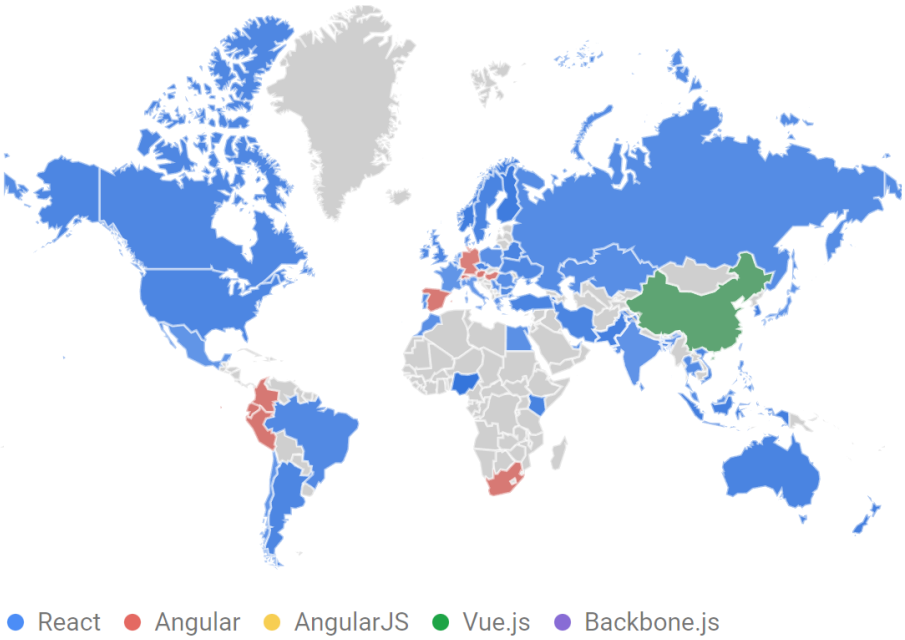

Abbildung 1.3: Google Trends Weltweiteverteilung von Suchanfragen nach Frameworks der letzten 5 Jahre [2]

2 Vue.js

In diesem Kapitel wird genauer auf Vue.js eingegangen. Dabei werden die Ziele von Vue.js erläutert. Weiterhin wird auf die Historie und Meilensteine in der Entwicklung von Vue.js eingegangen.

2.1 Was ist Vue.js?

Bei Vue.js oder einfach nur Vue genannt handelt es sich um ein progressives JavaScript-Frontend-Framework. Progressiv heißt in diesem Zusammenhang, dass es sich an die Bedürfnisse der Entwickler anpassen lässt. Der Kern von Vue.js kümmert sich ausschließlich um den View-Layer und ist rund 20 KB groß. Für weitere benötigte Features kann Vue.js modular erweitert werden. [6, S. 523-524]
Das Konzept von Vue.js ermöglicht es, Vue.js in bestehende Projekte zu integrieren und soll es leichter machen, das Framework eines bestehenden Projekts schrittweise durch Vue.js zu ersetzen [8, S. 1].

2.2 Geschichte

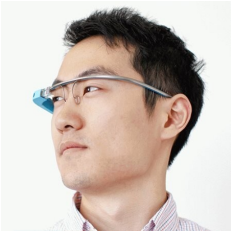

Abbildung 2.1: Evan You [10]

Erschaffer und Projektleiter von Vue.js ist der gebürtige Chinese Evan You (Abb. 2.1). Evan You arbeitete nach seinem Studium zunächst bei Google und später bei Meteor, wo er an Meteor.js beteiligt war [10].
Das Vue.js-Projekt startete er im Juli 2013 mit der Absicht *ein kleines Angular* zu schreiben. Während seiner Zeit bei Google war er bereits in vielen Projekten mit Angular involviert. Sein Ziel mit Vue.js war es zunächst, die seiner Meinung nach schlechten Designentscheidungen von Angular herauszufiltern und die seiner Ansicht nach guten Aspekte auf Vue.js

zu übertragen. Evan You entwickelte bei Google verschiedene Prototypen und stellte fest, dass Angular aufgrund seiner Komplexität für kleinere Projekte ungeeignet ist. Zudem ist das Framework zu starr, um schnell auf Änderungen und neue Anforderungen in einem Prototypen-Projekt zu reagieren. Öffentlich wurde Vue.js im Frühjahr 2014. Die erste Vollversion wurde im Oktober 2015 veröffentlicht. Bei der 2016 veröffentlichten Version v2.0 handelt es sich um eine von Grund auf Neuimplementierung. [9, 1:11-4:42]
Die aktuellste Version ist Vue 3 und erschien im Herbst 2020. Der Kern von Vue.js wurde dabei in TypeScript neu implementiert, was die Performance und den TypeScript-Support des Frameworks verbesserte. Neues Feature ist unter anderem die Composition API, dazu mehr im Abschnitt 3.1. [11]
Trotz der Inkompatibilität von Vue 2 und Vue 3 werden ein Großteil der Vue APIs zwischen diesen geteilt, was einen Umstieg erleichtert. [7]

3 Technische Details

In diesem Kapitel soll es um die technischen Details von Vue.js gehen. Es wird ein Überblick gezeigt von verschiedenen in Vue.js vorhandenen Elementen und es wird auf diese anhand von Beispielen genauer eingegangen. Die Betrachtung bezieht sich auf Vue 3. Aufgrund der Kürze der Arbeit und des Funktionsumfangs von Vue.js ist es nicht mögliche alle Features abzudecken. Das Kapitel dient dazu, dem Leser einen ersten Eindruck von Vue.js zu geben.

3.1 Options API vs. Composition API

Vue.js organisiert einzelne Komponenten als *Single-File Components (SFC)*, die im HTML-ähnlichen Dateiformat mit der Dateiendung `.vue` vorliegen. In einem SFC werden Aussehen, Struktur und Logik einer Komponente gebündelt, bestehend aus HTML-, CSS- und JavaScript-Elementen. Für den Aufbau einer solchen SFC gibt es seit Vue 3 zwei verschiedene Möglichkeiten. Zum einen gibt es die *Options API* und zum anderen die *Composition API*. [12]

Options API

Die länger vorhandene Variante ist die Options API. Bei der Options API wird für die Komponente ein JavaScript Objekt angelegt. Das Objekt kann verschiedene Optionen enthalten, darunter `data` für Daten, `methods` für Methoden und auch Lifecycle-hooks wie `mounted`. [12]

Composition API

Die Composition API kam mit Vue 3 hinzu und wurde später für Vue 2 nachgereicht [7]. Bei der Verwendung der Composition API werden in einem `import`-Statement die benötigten API-Features wie zum Beispiel Lifecycle-hooks angegeben. Der Code bei Verwendung der Composition API wird in der Regel zwischen einem `<script setup>` Tag verwendet. Durch die Verwendung des Tags wird zur Compilezeit eine Transformation durchgeführt, die es ermöglicht, die Composition API mit weniger redundantem Code zu nutzen. Somit ist es möglich, Variablen und Funktionen direkt im Template zu verwenden. [12]

Gegenüberstellung

```vue
<script>
export default {
  // reactive state
  data() {
    return {
     wasPressed : false
    }
  },

  // functions that mutate state and trigger updates
  methods: {
    setStatus() {
      this.wasPressed = true
    }
  },

  // lifecycle hooks
  mounted() {
    console.log(‘mounted‘)
  },

  update(){
    console.log(‘update‘)
  },

  unmounted(){
    console.log(‘unmounted‘)
  }
}
</script>

<template>
  <button @click="setStatus">Button was Pressed: {{ wasPressed }}</button>
</template>
```

Listing 3.1: Options API

```vue
<script setup>
import { ref, onMounted, onUpdated, onUnmounted } from ’vue’

// reactive state
const wasPressed = ref(false)

// functions that mutate state and trigger updates
function setStatus() {
  wasPressed.value = true;
}

// lifecycle hooks
onMounted(() => {
   console.log(‘mounted‘)
})

onUpdated(() => {
   console.log(‘updated‘)
})

 onUnmounted(() => {
   console.log(‘unmounted‘)
})

</script>

<template>
  <button @click="setStatus">Button was Pressed: {{ wasPressed }}</button>
</template>
```

Listing 3.2: Composition API

In Vue.js können beide APIs ohne Einschränkung genutzt werden. Hier sind zwei gleichwertige Beispiele in 3.1 und 3.2 dargestellt, die jeweils mit einer der APIs implementiert wurden. Es werden Veränderungen im Lifecycle auf der Konsole ausgegeben und es wird festgehalten, ob ein Button gedrückt wurde. Für Anfänger mit ersten Erfahrungen in Objektorientierung wird die Options API empfohlen. Ansonsten sollte die Options API nur bei weniger komplexen Kleinprojekten genutzt werden. Für größere Projekte wird die Composition API empfohlen. [12]

Fazit zu Options API vs. Composition API

Die Composition API ist deutlich übersichtlicher, strukturierter und spart redundanz im Code. Auch bei weniger komplexen Kleinprojekten könnte sich die Options API als ungeeignet erweisen, da unvorhergesehene Größenänderungen dazu führen können, dass die wartbare Composition API die bessere Wahl ist. Die Composition API ist der Options API vorzuziehen.

3.2 Direktiven in Vue.js

Direktiven beschreiben ein gewünschtes Verhalten in der View. In Vue.js beginnen diese Direktiven mit `v-`. Vue.js bietet einige vordefinierte Direktiven wie `v-for`, `v-if`, `v-else`, `v-show`, `v-on`, `v-slot`, `v-bind` und `v-model`, bietet aber auch die Möglichkeit, eigene Direktiven zu erstellen. [14, S. 10]
Es wird sich auf eine Auswahl der vordefinierten Direktiven beschränkt. Eine Liste aller Direktiven[1] und wie man eigene Direktiven[2] erstellt, ist in der Dokumentation von Vue.js zu finden. Auf Direktiven zur Datenbindung wird an passenderer Stelle in Abschnitt 3.4 eingegangen.

v-for

`v-for` funktioniert wie eine foreach-Schleife. Die `v-for`-Direktive ermöglicht es über eine Sammlung von Werten zu iterieren und die Daten in der View zu nutzen. Erlaubte Datentypen sind `Array`, `Object`, `number`, `string` und solche, die das Interface `Iterable` implementieren. [13]

```
1 <div v-for="element in list">
2   {{ element.value }}
3 </div>
```

Listing 3.3: `v-for`-Direktive

v-if

Die Direktive `v-if` ähnelt in der Funktionsweise einem if-Statement in der Programmierung. `v-if` prüft anhand des übergebenen Wahrheitswerts, ob ein HTML-Element angezeigt werden soll. Mit der Verwendung von `v-if` können Alternativen mit `v-else` für den Fall, dass die Bedingung nicht erfüllt ist, oder mit `v-else-if` für eine bedingte Alternative ergänzt werden. Entsprechend dem Wahrheitswert werden dann die Alternativen angezeigt. [13]

[1] https://vuejs.org/api/built-in-directives.html
[2] https://vuejs.org/guide/reusability/custom-directives.html

```
<div v-if="condition">
  condition is true
</div>
<div v-else-if="anotherCondition">
  anotherCondition is true
</div>
<div v-else>
  condition and anotherCondition are false
</div>
```

Listing 3.4: `v-if`-Direktive

v-on

Mit der `v-on`-Direktive wird ein Event-Listener an das HTML-Element gebunden. Die Direktive erhält als Argument das Event, auf das der Listener reagieren soll, sowie die darauf auszuführende Aktion. Als Kurzform von `v-on` ist auch `@` möglich. Mit `$event` ist es möglich Eventeigenschaften weiter zugeben. Darüber hinaus gibt es noch Modifier, die das zu erwartende Event präzisieren. [13]

```
<button v-on:click="buttonClicked"></button>
<button @click="buttonClickedButShortHand"></button>
<button v-on:click.right="buttonRightClicked"></button>
```

Listing 3.5: `v-on`-Direktive

3.3 Komponenten in Vue.js

Komponenten sind wiederverwendbare Bausteine. Die Komponenten können beliebig oft in der View verwendet werden und im Model unabhängig angesteuert werden. Vue.js bietet die Möglichkeit, Komponenten selbst aus HTML, CSS, sowie JavaScript zu definieren und sie später mit einem selbst definierten HTML-Tag anzusteuern. [14, S. 11-12]
Wie in Abschnitt 3.1 bereits erwähnt, werden in Vue.js Komponenten normalerweise als SFC geschrieben. Eine Komponente kann wie in Abschnitt 3.1 erwähnt mit Composition API oder Options API erstellt werden. Alternativ kann man die Komponente als JavaScript Objekt erstellen, um den Build-Schritt zu umgehen (siehe Listing 3.6). Eine Komponente kann dann mit einem Alias im Script-Teile einer anderen Komponente importiert werden und muss dort noch als Komponente registriert werden. Nach dem Registrieren kann der Alias als Tag für die importierte Komponente verwendet werden. [15]

```
export default {
  data() {
    return {
     wasPressed : false
    }
  },
  template:`<button @click="setStatus">Button was Pressed: {{ wasPressed }}</button>`
}
```

Listing 3.6: Komponente als JavaScript Objekt

```
<script>

import ButtonStatusChecker from './ButtonStatusChecker.vue' //import of component

export default {
  components: {
    ButtonStatusChecker //component registration
  }
}
</script>

<template>
  <ButtonStatusChecker /> //using component
</template>
```

Listing 3.7: Verwendung einer Komponente

Neben der lokalen Registrierung existiert die Möglichkeit eine Komponente global zu registrieren, um sie nicht bei jeder Verwendung in einer anderen Komponente extra importieren und registrieren zu müssen. Die globale Registrierung ist mit Aufruf der `component()`-Methode der App-Instanz möglich. Der `component()`-Methode wird dann der Alias und die importierte `.vue`-Datei übergeben. Alternativ kann auch die Implementierung beim Aufruf direkt übergeben werden. [16]

```
import { createApp } from 'vue'
import ButtonStatusChecker from './ButtonStatusChecker.vue'
const app = createApp({})
app.component('ButtonStatusChecker', ButtonStatusChecker)
```

Listing 3.8: Globale Registrierung einer Komponente

Um Daten an eine Komponente zu übergeben, gibt es die `props`-Option beim Erstellen einer Komponente. Das `props`-Attribut erhält einen Namen, über die es dann später in HTML angesteuert werden kann. [15]

```
<script>
export default {
  props: ['text']
}
</script>

<template>
  <p>{{ text }}<p>
</template>
```

Listing 3.9: Erstellung eines `props`

```
<Paragraph text="Hallo Welt" />
```

Listing 3.10: Nutzung eines `props`

Abgesehen von den bereits aufgezählten Features, bietet Vue.js für Komponenten weitere Features wie die Möglichkeit auf Events zu reagieren, die Verwendung dynamischer Komponenten und Inhalte mittels `<slot>` einer Komponente zu übergeben. Eine Anleitung zur Verwendung dieser Features ist in der Vue.js Dokumentation nachzulesen[3]. [15]

3.4 Datenbindung in Vue.js

Das Binding ist einer der Hauptaufgaben eines Frontend Frameworks. Beim Binding geht es darum, Daten von View und Model aneinander zubinden. [14, S. 11]

Mustache-Syntax

Die einfachste Form der Datenbindung in Vue.js ist mit der auch aus anderen Frameworks bekannten Mustache-Syntax möglich. Bei der Mustache-Syntax wird ein Variablenname aus dem Modell in der Komponente zwischen zwei geschweiften Klammern gesetzt. Angezeigt wird dann die aktuelle String-Repräsentation des Wertes. Die Mustache-Syntax ist unidirektional vom Model zur View. [17]

```
<script>
export default {
  data() {
    return {
     text : 'Hallo Welt'
    }
  }
</script>

<template>
  <p>{{ text }}</p>
</template>
```

Listing 3.11: Mustache-Syntax

[3] https://vuejs.org/guide/essentials/component-basics.html

v-text

Neben den in Abschnitt 3.2 erwähnten Direktiven gibt es noch weitere Direktiven zur Datenbindung. Die `v-text`-Direktive ist eine Alternative zur Mustache-Syntax, diese überschreibt jedoch den gesamten existierenden Inhalt des HTML-Elements bei einer Änderung im Model. Eine Änderung in Teilen wie bei der Mustache-Syntax ist nicht möglich. [13]

```
<script>
export default {
  data() {
    return {
     text : 'Hallo Welt'
    }
  }
</script>

<template>
  <p v-text="text"></p>
</template>
```

Listing 3.12: `v-text`-Direktive

v-html

Die `v-html`-Direktive erlaubt es, Daten aus dem Model in der View als HTML interpretieren zu lassen. Aus Sicherheitsgründen empfiehlt es sich nur HTML-Code aus vertrauenswürdigen Quellen zu nutzen. [17]

```
<script>
export default {
  data() {
    return {
     heading : '<h1>Hallo Welt</h1>'
    }
  }
</script>

<template>
  <p v-html="heading"></p>
</template>
```

Listing 3.13: `v-html`-Direktive

v-bind

Um Daten an HTML-Attribute zubinden, gibt es die `v-bind`-Direktive. Da `v-bind` eine häufig genutzte Direktive ist, gibt es eine Kurzform, die aus einem vorangestellten Doppelpunkt besteht. [17]

```
<script>
export default {
  data() {
    return {
     id : 'greeting',
     idShortHand : 'target',
     isButtonOn: false
    }
  }
</script>

<template>
  <p v-bind:id="id">Hallo</p>
  <p :id="idShortHand">Welt</p>
  <button :disabled="isButtonOn">Drück mich</button>
</template>
```

Listing 3.14: `v-bind`-Direktive

v-model

Die bisher vorgestellten Binding Varianten sind unidirektional, für bidirektionales Binding gibt es die `v-model`-Direktive. Die `v-model`-Direktive ist eine Verbindung aus `v-on`-Direktive und `v-bind`-Direktive. Mit der `v-bind`-Direktive ist es zum Beispiel möglich, vorausgefüllte Inputfelder zu erstellen. Bei Eingabe in das Inputfeld wird entsprechend das Datenfeld im Modell aktualisiert und bei Änderungen im Modell wird der Text im Inputfeld aktualisiert. [18]

```
<script>
export default {
  data() {
    return {
     username : 'Mustermann',
    }
  }
</script>

<template>
  <input v-model="username" />
</template>
```

Listing 3.15: `v-model`-Direktive

3.5 Routing in Vue.js

Für das Routing muss man zwischen Multi-Page und Single-Page Anwendungen unterscheiden. Bei einer Multi-Page Anwendung (MPA) erfolgt das Aufrufen einer neuen Unterseite über die Kommunikation mit dem Server, die Seite wird dann serverseitig gerendert. Bei MPA handelt es sich um einen älteren Ansatz.[19, S. 262]
In Vue.js wird der Ansatz der Single-Page Anwendung (SPA) verfolgt. Bei Single-Page Anwendungen existieren Unterseiten nur simuliert und die Inhalte einer einzelnen Seite werden mithilfe von JavaScript ausgetauscht. [8, S. 174-175]

Installation

Der offizielle Router von Vue.js ist nicht Teil des Vue-Kerns und muss daher bei Bedarf gesondert installiert werden. Die aktuelle mit Vue 3 kompatible Router-Version ist Version 4 und kann über npm mit dem Befehl `npm install vue-router@4` installiert werden.[20]

Router initialisieren

Man muss den Router zunächst erstellen. Dem Router werden die Routen, bestehend aus Pfad und aufzurufender Komponente übergeben. Zusätzlich wird beim Erstellen des Routers angegeben, wie die Routinghistorie festgehalten werden soll. Abschließend muss der Vue-App-Instanz mitgeteilt werden, dass sie den Router nutzen soll. [21]

```
1  //import of components
2  import Home from './Home.vue'
3  import About from './About.vue'
4
5  //define the routes
6  const routes = [
7    { path: '/', component: Home },
8    { path: '/about', component: About },
9  ]
10
11 //create the router instance
12 const router = VueRouter.createRouter({
13   history: VueRouter.createWebHashHistory(), //history mode
14   routes: routes,
15 })
16
17 // creat app instance
18 const app = Vue.createApp({})
19 app.use(router)
20 app.mount('#app')
```

Listing 3.16: Router initialisieren

Verwendung des Routers

Um den Router zu verwenden und auf eine neue Unterseite zu verlinken, gibt es den HTML-Tag `router-link`, mit dem Attribut `to` kann dann der gewünschte Pfad angegeben werden. Wie bereits erwähnt, verwendet man Vue.js in der Regel für SPA, in denen ein Seitenwechsel nur simuliert stattfindet. Mit dem HTML-Tag `router-view` gibt man den Bereich der Seite an, in dem beim Aufrufen eines Links die neue Komponente gerendert werden soll. Um diesen `router-view`-Tag herum können dann Elemente, die immer zu sehen sein sollen, angelegt werden. [21]

```
<div class="navbar">
  <router-link to="/">Home</router-link>
  <router-link to="/about">About</router-link>
<\div>
<router-view></router-view>
```

Listing 3.17: Verwendung des Routers

Routing mit Parametern

In Vue.js kann man Daten an eine aufgerufene Komponente durch Verwendung von Parametern im Pfad übergeben. Um einen Parameter zu verwenden, muss man diesen beim Erstellen der Routen mit vorangestelltem : angeben. In der Komponente kann auf den Parameter wie in Listing 3.19 zugegriffen werden. [22]

```
const routes = [
  { path: '/cars/:id', component: Cars },
]
```

Listing 3.18: Route mit Parameter

```
$route.params.id
```

Listing 3.19: Zugriff auf Routingparameter

Weitere Routing-Features

Weitere Features, die in Vue.js mithilfe des Routers genutzt werden können, sind unter anderem die Verwendung von Nested Routes[4] und das Ändern des History-Modes[5] beim Erstellen des Routers. Diese Features und weitere Features sind ausführlich in der Dokumentation des Vue-Routers beschrieben.

[4] https://router.vuejs.org/guide/essentials/nested-routes.html
[5] https://router.vuejs.org/guide/essentials/history-mode.html

3.6 State Management in Vue.js

In Vue.js besitzt jede Komponente ihren eigenen State, sowie eine View und Actions.

```
<script>
export default {
  // state
  data() {
    return {
     wasPressed : false
    }
  },

  // actions
  methods: {
    setStatus() {
      this.wasPressed = true
    }
  }
}
</script>

//view
<template>
  <p> {{ wasPressed }}</p>
</template>
```

Listing 3.20: State, View und Actions

Der Begriff *State* bezieht sich auf den Zustand von Daten und somit auf ihren aktuellen Wert. Möchte man von aus verschiedenen Komponenten auf denselben State zugreifen, biete Vue.js mit der Reactivity API eine Möglichkeit für einfaches State Management. Mit der Reactivity API kann man einen Store anlegen, welcher den State enthält. Der Store kann dann in den Komponenten, wo er benötigt wird, importiert werden. Aus Gründen der Wartbarkeit des Codes empfiehlt es sich nicht den State lokal in einer Komponente anzupassen, sondern im Store eine zugehörige Action anzulegen. [23]

```
import { reactive } from 'vue'

export const store = reactive({
  wasPressed : false,
  setStatus() {
      this.wasPressed = true
  }
})
```

Listing 3.21: Anlegen eines Stores mit Reactivity API

```
<template>
  <button @click="store.setStatus()">
    Button was Pressed: {{ store.wasPressed }}
  </button>
</template>
```

Listing 3.22: Vewendung des Stores

Für komplexeres State Management wird die offizielle State Management Library *Pinia* empfohlen. Pinia bietet unter anderem die Vorteile serverseitiges Rendering zu unterstützen und ein besseres Debugging durch eine Integration in die Vue-DevTools zu ermöglichen. Pinia löste Vuex als offizielle State Management Library ab und bietet unter anderem gegenüber von Vuex die Möglichkeit mehrere Stores zu haben sowie eine einfachere Anwendung. [23]

3.7 Architektur

Projektstruktur

Bei der Verwendung des Vue-CLI wird die Dateistruktur des Projekts bereits automatisch erstellt. Während der Initiierung hat man die Möglichkeit Erweiterungen wie den Router 3.5 und Pinia 3.6 hinzuzufügen. Fügt man die Erweiterungen hinzu, werden automatisch passende Ordner wie `router` und `stores` angelegt. Das Folder-by-Type-Prinzip ist die Standardstruktur für ein Vue-Projekt. Für den Komponenten-Ordner wird empfohlen, keine Unterverzeichnisse zu erstellen und die Komponenten auf einer Ebene zu belassen. [24]

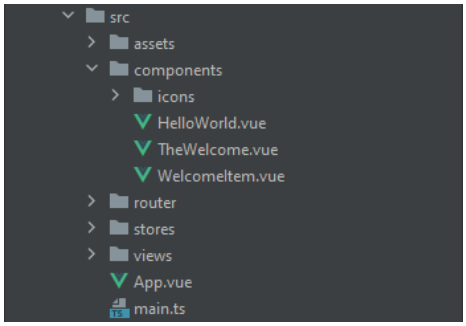

Abbildung 3.1: Projektstruktur

Der virtuelle DOM

Das Ändern des HTML-DOMs ist aufwändig und kann zu Performanceproblemen führen. Für eine bessere Performance benutzt Vue.js einen virtuellen DOM. Der virtuelle DOM ist eine Kopie des eigentlichen DOM. Änderungen werden zunächst am virtuellen DOM vorgenommen. Die Änderungen am virtuellen DOM ermöglicht es mehrere Änderungen zu bündeln, bevor der reelle DOM angepasst wird. [14, S. 10-11]

Model-View-ViewModel

In Vue.js findet das MVVM-Pattern Anwendung. Das sogenannte *ViewModel* übernimmt die Vermittlung zwischen der View und der Daten mit Businesslogik im Model. Die Aufgabe des ViewModels übernimmt Vue.js.[14, S. 43][6]

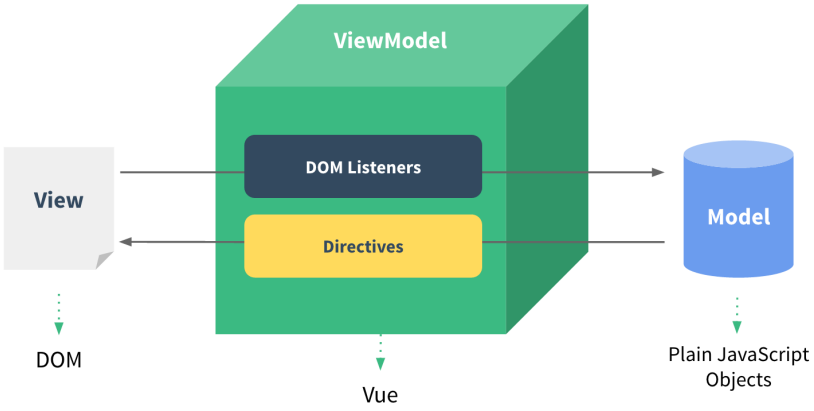

Abbildung 3.2: MVVM in Vue.js [25]

[6] In der Quelle ist die Rede von "MVVC". Aus dem Kontext ist klar, dass es sich um einen Tippfehler handelt und "MVVM"gemeint ist.

4 Vergleich zu Angular anhand einer einfachen App

4.1 Einführung in den Vergleich

Für eine Gegenüberstellung mit Angular wurde jeweils in Vue.js und in Angular eine simple Einkaufslistenapp erstellt. Das Ziel war es, mit beiden Frameworks eine möglichst ähnliche Einkaufslistenapp zu entwickeln und so von beiden Frameworks einen direkten Vergleich zu erstellen. Die App besteht aus zwei Seiten: einer Checklisten-Seite und einer Seite zur Erstellung und Bearbeitung der Einträge.

Die Checklisten-Seite enthält eine Liste mit Mengenangaben der einzukaufenden Artikel und ermöglicht es dem Benutzer, die Artikel abzuhaken. Auf der Bearbeiten-Seite können Artikel mit Mengenangaben erstellt, bearbeitet und gelöscht werden. Für diese Gegenüberstellung wurde auf ein Backend und auf aufwändiges Design verzichtet. Ein Screenshot der beiden Seiten (siehe Abbildung 1 und 2) und die Links zu den GitHub Repositories der beiden Apps befinden sich im Anhang.

4.2 Gegenüberstellung von Vue.js und Angular anhand einer einfachen App

Zunächst wird auf die Ähnlichkeiten zwischen beiden Frameworks eingegangen, die beim Entwickeln der Einkaufslistenapp aufgefallen sind. Beide Frameworks haben ähnliche Direktiven wie zum Beispiel `v-model` und `[(ngModel)]` oder `v-for` und `*ngFor`, die in ihrer Verwendung fast gleich sind. Die beiden Frameworks ermöglichen unidirektionale Datenbindung mit der Mustache-Syntax. Das Routing ist bei beiden Frameworks nicht standardmäßig vorhanden. Bei Angular muss das Routing erst aktiviert werden, in Vue.js muss die Routing-Erweiterung installiert werden. Die Verwendung des Routers ist bei beiden ähnlich. Man legt den Pfad mit der zugehörigen Komponente in einem Array fest und übergibt dies abschließend der App-Instanz. Mit einem `<router-outlet>`-Tag in Angular oder einem `<RouterView>`-Tag in Vue.js wird dann angegeben, wo die aktuelle Route gerendert werden soll. Sowohl Vue.js als auch Angular bieten bereits beim Erstellen des Projekts die Möglichkeit, das Routing zu ergänzen. Während in Vue.js in der Regel mit SFC gearbeitet wird, besteht in Angular eine Komponente aus vier Dateien. Die vier Dateien einer Angular Komponente unterteilen sich in eine TypeScript-, eine HTML-, eine Style- und einer Verwaltungsdatei. In Vue.js legt man für eine neue Komponente eine neue `.vue`-Datei an,

in Angular erfolgt es über die Angular CLI, welche einen Ordner mit den vier Dateien generiert. Für die Einkauflistenapp sollte der Inhalt der Liste zentral sein, damit sowohl von der Checklisten-Seite als auch von der Bearbeiten-Seite auf die Einträge zugegriffen werden kann. In Vue.js ist dies mit dem in Abschnitt 3.6 erwähnten Pinia möglich. Mit Pinia wurde eine Store für die Einkaufliste mit den darin enthaltenen Items erstellt. Auf den Einkaufslisten-Store kann dann von der Checklisten-Seite und der Bearbeiten-Seite zugegriffen werden. In Angular wurde es mit einem Service gelöst. Der erstellte Service in Angular ermöglicht einen gleichwertigen zentralen Zugriff.
Auffällig ist der Größenunterschied der beiden Projektordner. Die Implementierung mit Vue.js hat 29 MB, dagegen hat die Implementierung mit Angular eine Größe von 370 MB. Der Entwicklungsserver der Vue.js Implementierung braucht 127 MB Arbeitsspeicher, für den Entwicklungsserver der Angular Implementierung werden 460 MB Arbeitsspeicher beansprucht. Nach dem Build-Prozess beansprucht die Vue.js-Implementierung einen Speicherplatz von 88 KB und einen Arbeitsspeicher von 84 MB. Die Angular-Implementierung erfordert nach dem Build-Prozess einen Speicherplatz von 287 KB und einen Arbeitsspeicher von 92 MB.

5 Fazit

Die in Abschnitt 2.2 von Evan You definierten Ziele für Vue.js sind erreicht. Die Absicht ein schlankes Framework zu erstellen ist erfüllt und wird deutlich beim Vergleichen der Speicherauslastung der zwei äquivalenten Einkaufslistenapps im Entwicklungszustand. Nach Durchführen des Builds fällt die Schlankheit von Vue.js weniger auf, wie sich diese bei größeren Apps verhält lässt sich aus dem Versuch nicht ableiten. Im fertigen Build der Vue.js-Implementierung ist die Belegung des Arbeitsspeichers leicht geringer als die der Angular-Implementierung. Der fertige Build der Vue.js-Implementierung benötigt rund 1/3 des Festplattenspeichers von Angular, auch wenn der benötigte Festplattenspeicher der Angular-Implementierung mit unter 300 KB immer noch sehr gering ist. Die Dynamik von Vue.js wird durch die modulare Erweiterbarkeit und durch das Konzept der SFC deutlich. Während man in Angular immer wieder zwischen drei Dateien springen muss, um eine Komponente zu bearbeiten, ist in Vue.js alles zentral in einer Datei. Die ähnliche Umsetzung einiger Konzepte wie das Routing oder die Verwendung von Direktiven machen den Wechsel zwischen Angular und Vue.js einfacher. Für kleinere Anwendungen ist Vue.js aufgrund der bereits genannten Konzepte vorzuziehen. Wie sich Vue.js im Vergleich zu Angular bei größeren Projekten verhält, lässt sich aus dem Vergleich in Abschnitt 4 nicht eindeutig sagen. Der Vergleich in Abschnitt 4 ist für Rückschlüsse auf umfangreichere Projekte ungeeignet, da eine Betrachtung von Aspekten wie einem Zusammenspiel mit einem Backend oder Betrachtung des Featureumfangs fehlen. Der Vergleich in Abschnitt 4 bringt vielmehr einen Einblick von Vue.js für Entwickler, die bereits Erfahrung mit Angular haben. Abschließend lässt sich sagen, dass Vue.js die definierten Anforderungen umsetzt und eine schlankere Alternative zu Angular darstellt.

6 Ausblick

Es ist sinnvoll einen noch passenderen Versuch zu finden, um zu testen, wie sich Vue.js im Vergleich zu Angular bei größeren Projekten schlägt. Des Weiteren wäre ein Vergleich zu React.js mit einer ähnlichen Methode wie in Abschnitt 4 möglich. Darüber hinaus bietet Vue.js weitere Features, die aufgrund der Kürze der Arbeit keine Erwähnung fanden. Features, die man noch betrachten könnte, wären Teleport, serverseitiges Rendern, Form Validation und vieles mehr. Was in dieser Arbeit nicht betrachtet wurde, sind UI Frameworks wie Vuetify[1] in Verbindung mit Vue.js.
Der Support für Vue 2 endet am 31. Dezember 2023 [7]. Weitere Pläne für Vue.js werden gegebenenfalls auf der VueConfUS bekannt gemacht, welche vom 24. bis 26. Mai 2023 stattfinden wird.[2]

[1] https://vuetifyjs.com/en/
[2] http://vueconf.us/

Literaturverzeichnis

[1] Thomas Sigdestad *Front-end frameworks: What is important right now?*, https://enonic.com/blog/front-end-frameworks-what-is-important, (abgerufen 26.04.2023)

[2] — *Google Trends JavaScript Frameworks*, https://trends.google.de/trends/explore/GEO_MAP/1682502600?hl=de&tz=-120&date=today+5-y&hl=de&q=%2Fm%2F012l1vxv,%2Fg%2F11c6w0ddw9,%2Fm%2F0j45p7w,%2Fg%2F11c0vmgx5d,%2Fm%2F0h94450&sni=3, (abgerufen 26.04.2023)

[3] — *Stackoverflow Statistik zur Häufigkeit von Fragen nach Framework1*, https://insights.stackoverflow.com/trends?tags=reactjs%2Cangular%2Cvue.js%2Cember.js%2Cbackbone.js%2Cknockout.js%2Cangularjshttps://insights.stackoverflow.com/trends?tags=reactjs%2Cangular%2Cvue.js%2Cember.js%2Cbackbone.js%2Cknockout.js%2Cangularjs, (abgerufen 26.04.2023)

[4] Cory Gackenheimer *Introduction to React.*, Apress, 2015.

[5] — *React The library for web and native user interfaces*, https://react.dev/, (abgerufen 27.04.2023)

[6] Sufyan bin Uzayr, Nicholas Cloud , Tim Ambler *JavaScript Frameworks for Modern Web Development* Springer, 2019

[7] — *Frequently Asked Questions | Vue.js*, https://vuejs.org/about/faq.html, (abgerufen 28.04.2023)

[8] Lars Peterke *Vue.js kurz & gut*, O'REILLY, 2019

[9] — *VueNYC - Vue.js: the Progressive Framework - Evan You*, https://www.youtube.com/watch?v=p2P3z7p_zTI, (abgerufen 01.05.2023)

[10] — *Evan You Creator of Vue.js* https://www.linkedin.com/in/evanyou (abgerufen 01.05.2023)

[11] Antony Konstantinidis *Vue.js 3 – das JavaScript Framework im neuem Gewand* https://vuejs.de/artikel/vuejs-3-release/ (abgerufen 02.05.2023)

[12] — *Introduction | Vue.js* https://vuejs.org/guide/introduction.html (abgerufen 05.05.2023)

[13] — *Built-in Directives | Vue.js* https://vuejs.org/api/built-in-directives.html (abgerufen 05.05.2023)

[14] Ralph Steyer *Webanwendungen erstellen mit Vue.js*, Springer, 2019

[15] — *Components Basics | Vue.js* https://vuejs.org/guide/essentials/component-basics.html (abgerufen 06.05.2023)

[16] — *Component Registration | Vue.js* https://vuejs.org/guide/components/registration.html (abgerufen 06.05.2023)

[17] — *Template Syntax | Vue.js* https://vuejs.org/guide/essentials/template-syntax.html (abgerufen 09.05.2023)

[18] — *Component v-model | Vue.js* https://vuejs.org/guide/components/v-model.html (abgerufen 09.05.2023)

[19] Marin Kaluža,Krešimir Troskot, Bernard Vukelić *Comparison of front-end frameworks for web applications development* Zbornik Veleučilište u Rijeci, 2018

[20] — *Installation | Vue Router* https://router.vuejs.org/installation.html (abgerufen 10.05.2023)

[21] — *Getting Started | Vue Router* https://router.vuejs.org/guide/ (abgerufen 10.05.2023)

[22] — *Dynamic Route Matching with Params| Vue Router* https://router.vuejs.org/guide/essentials/dynamic-matching.html (abgerufen 10.05.2023)

[23] — *State Management | Vue.js* https://vuejs.org/guide/scaling-up/state-management.html (abgerufen 11.05.2023)

[24] Daniel Kelly *How to Structure a Large Scale Vue.js Application - Vue School Articles* https://vueschool.io/articles/vuejs-tutorials/how-to-structure-a-large-scale-vue-js-application/ (abgerufen 12.05.2023)

[25] — *Getting Started - vue.js* https://012.vuejs.org/guide/ (abgerufen 12.05.2023)

Anhang

Einkaufslisten-App

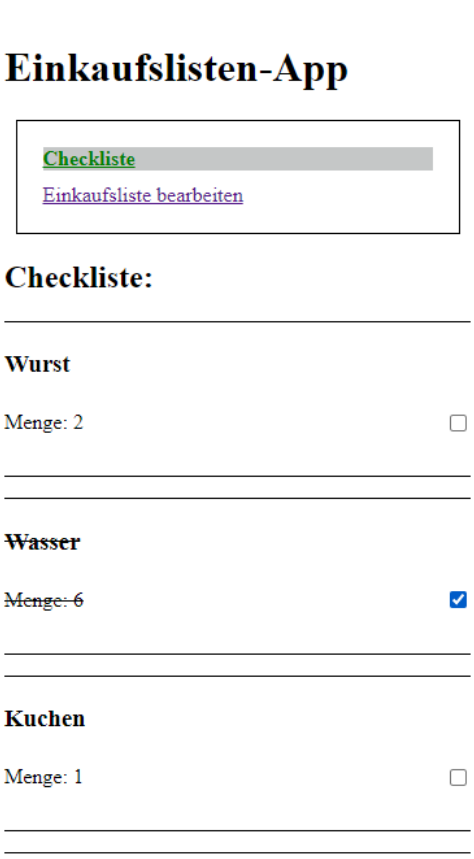

Abbildung 1: Checklistenseite der Einkaufslisten-App

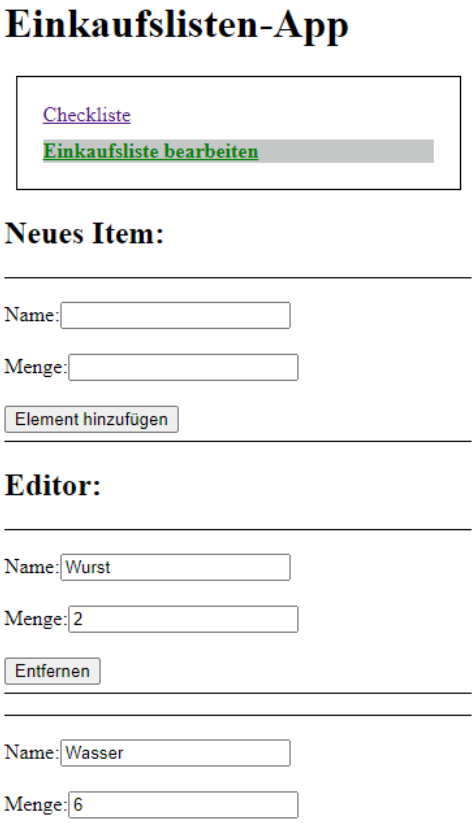

Abbildung 2: Bearbeiten- und Erstellenseite der Einkaufslisten-App

Implementierung mit Angular: https://github.com/mbbl33/angular-einkaufsliste

Implementierung mit Vue.js: https://github.com/mbbl33/vue-einkaufsliste