Hauptseminar-Arbeit

Vue.js

Prüfungsleitung des Moduls
CS1025 Hauptseminar

von

Maximilian Biebl
`Matrikelnr.: 5323481`

am 10. 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, 10. Mai 2023

Zusammenfassung

Heutzutage ist es Standard, dass man mit einer Webseite interagieren kann und sich die Inhalte der Webseite dynamisch anpassen. 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 gängigsten dieser Frameworks sind React, Angular und Vue.js. Diese Frontend-Frameworks bilden das Bindeglied zwischen der Logik in JavaScript und der Darstellung in HTML und CSS. In dieser Ausarbeitung möchten wir uns genauer mit dem Framework Vue.js beschäftigen und einen Vergleich zu React und Angular ziehen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Frontend Frameworks

In diesem Kapitel möchte ich zunächst allgemein in die JavaScript-Frontend-Frameworks einführen. 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 Benutzer gestellt. Die Entwickler einer solchen Webapplikation erwarten eine einheitliche Codequalität und einheitliche Strukturen für eine bessere Wartbarkeit der Webapplikation. Um beim Erstellen neuer Webapplikationen den 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. [2]
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.

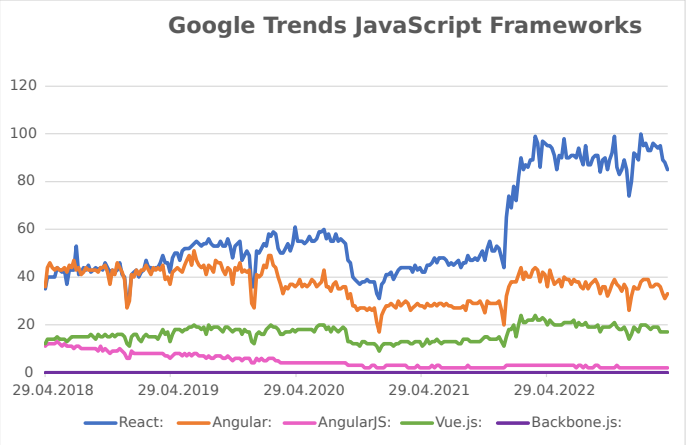

Abbildung 1.1: Google Trends zur Häufigkeit von Suchanfragen nach Frameworks der letzten 5 Jahre [3]

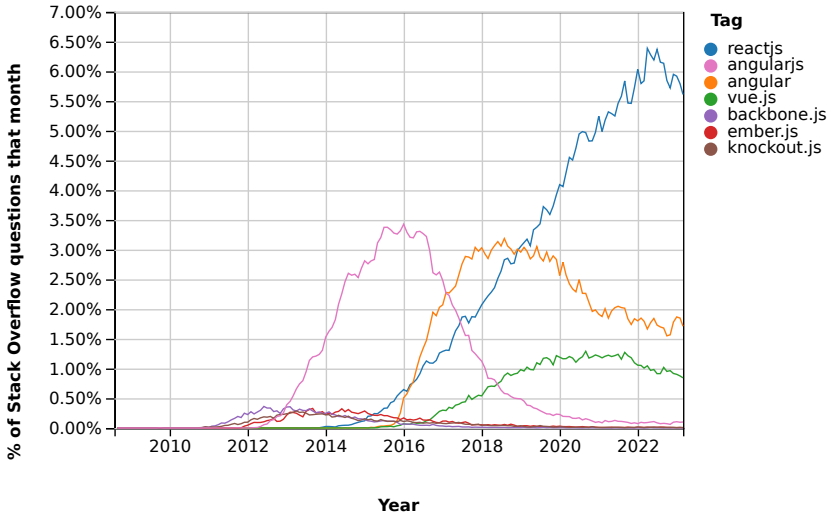

Abbildung 1.2: Stack Overflow Statistik zur Häufigkeit von Fragen nach Framework [4]

1.1 Vue.js

Vue.js wurde von Evan You als Nebenprojekt entwickelt und erschien 2014. Das Projekt finanziert sich laut eigenen Angaben aus Spenden und wird von sowohl Vollzeitentwicklern und Freiwilligen gepflegt [8]. Nach Google Trends und Stack Overflow Statistik ist es aktuell das drittbeliebteste Frontend Framework [3] [4]. 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 [7, S. 523].

1.2 Angular

Angular wurde von Google entwickelt und erschien 2016 ursprünglich als *AngularJs 2.0*, dabei 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 [7, S. 209-210]. Nach Google Trends und Stack Overflow Statistik ist es aktuell das zweitbeliebteste Frontend Framework [3] [4].

1.3 React.js

Das laut Stack Overflow Statistik und Google Trends gefragteste Frontend Framework ist aktuell React.js [3] [4]. 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 sind [5, S. 1]. Mittlerweile ist es ein Open-Source-Projekt des Facebookmutterkonzerns Meta. React setzt dabei auf *JavaScript syntax extension* (JSX), was einen hybriden Code aus JavaScript und HTML-Elementen ermöglicht. [6]

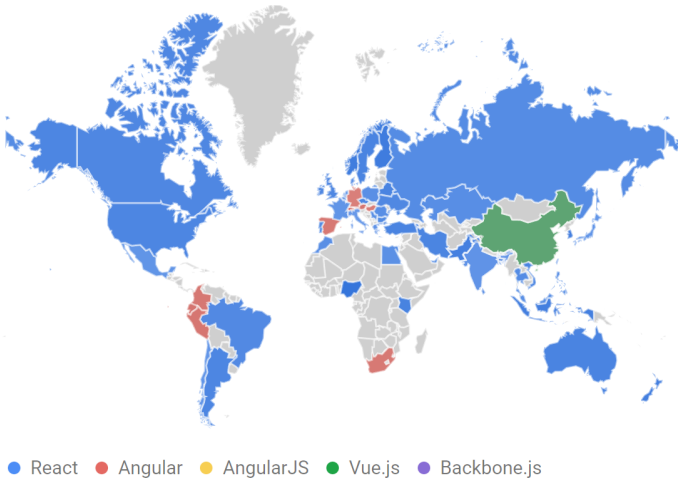

Abbildung 1.3: Googlge Trends Weltweiteverteilung von Suchanfragen nach Frameworks der letzten 5 Jahre [3]

2 Vue.js

In diesem Kapitel möchte ich genauer auf Vue.js eingehen. Dabei möchte ich Vue.js und seine Ziele erläutern. Weiterhin möchte ich auf die Historie und Meilensteine in der Entwicklung von Vue.js eingehen.

2.1 Was ist Vue.js?

Bei Vue.js oder einfach nur Vue genannt handelt es sich um ein progressives JavaScript Frontend Framework. Progressive heißt in diesem Zusammenhang, dass es sich an die Bedürfnisse der Entwickler anpassen lässt. Der Kern von Vue kümmert sich ausschließlich um den View-Layer und ist rund 20 KB groß. Für weitere benötigte Features kann Vue.js modular erweitert werden. [7, S. 523-524]
Das Konzept von Vue ermöglicht es, Vue in bestehende Projekte zu integrieren und soll es leichter machen, ein Projekt schrittweise von anderen Frameworks zu Vue.js migrieren zulassen [9, S. 1].

2.2 Geschichte

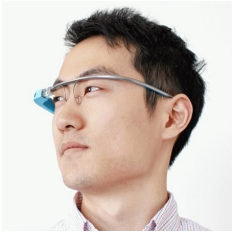

Abbildung 2.1: Evan You [11]

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 [11].
Das Vue.js-Projekt startete er im Juli 2013 mit der Absicht *ein kleines Angular* zuschreiben. Während seiner Zeit bei Google hatte er bereits viele Projekte mit Angular. Sein Ziel mit Vue war es zunächst, die seiner Meinung nach schlechten Sachen von Angular herauszufiltern und die seiner Ansicht nach guten Dinge auf Vue 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 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. [10, 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. [12]
Weitere neue Features sind Suspense, Teleport und die Möglichkeit mehrere Root-Elemente pro Template zuhaben. Trotz der Inkompatibilität von Vue 2 und Vue 3 werden ein Großteil der Vue APIs zwischen diesen geteilt, was einen Umstieg erleichtert. [8]

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. Ich beziehe mich dabei auf Vue 3.

3.1 Options API vs. Composition API

Vue 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*. [13]

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`. [13]

Composition API

Die Composition API kam mit Vue 3 hinzu und wurde später für Vue 2 nachgereicht [8]. Bei der Verwendung der Composition API wird 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. Mit dem Tag wird angegeben, dass zur Compilezeite eine Transformation durch geführt werden soll, die Composition API mit weniger Redundanz im Code zu nutzen. So können Variablen und Funktionen direkt im Template genutzt werden. [13]

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 können beide APIs ohne Einschränkung genutzt werden. In 3.1 und 3.2 sehen wir zwei gleichwertige Beispiele unterverwendung der unterschiedlichen APIs. Es werden Veränderungen im Lifecyle 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. [13]

Persönliche Meinung zu Options API vs. Composition API

Ich persönliche würde die Composition API der Options API vorziehen. Meine Begründung dafür ist, dass ich die Composition API deutlich übersichtlicher und strukturierter finde. Auch für weniger komplexe Kleinprojekte finde ich die Options API ungeeignet, da ein Projekt schnell größer werden kann als ursprünglich geplant und somit die wartbarere Composition API sinnvoll ist.

3.2 Direktiven in Vue.js

Direktiven beschreiben ein gewünschtes Verhalten in der View. In Vue beginnen diese Direktiven mit *v-*. Vue 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. [15, S. 10]
Ich möchte mich hier auf ein paar der vordefinierten Direktiven beschränken. Eine Liste aller Direktiven[1] und wie man eigene Direktiven[2] erstellt sind in der Dokumentation von Vue zu finden. Auf Direktiven zur Datenbindung gehe ich an passenderer Stelle in Abschnitt 3.4 ein.

v-for

v-for funktioniert wie eine klassische 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. [14]

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

Listing 3.3: *v-for*-Direktive

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

v-if

Die Direktive *v-if* funktioniert ähnlich der eines klassischen if-Statements 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 werden dann die Alternativen angezeigt. [14]

```
<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. Als Argument bekommt die Direktive das Event, auf welches der Listener warten soll und die anschließend 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. [14]

```
<button v-on:click="buttonClicked"></button>
<button @click="buttonClickedButShortHand"></button>
<button v-on:click.richt="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 bietet die Möglichkeit Komponenten selbst zu definieren aus HTML, CSS, sowie JavaScript und sie später mit einem eigen definierten HTML-Tag anzusteuern. [15, S. 11-12]
Wie in Abschnitt 3.1 bereits erwähnt, werden in Vue Komponenten normalerweise als SFC geschrieben. Eine Komponente kann wie in Abschnitt 3.1 erwähnt und wie in Listings 3.2 oder 3.1 erstellt werden. Alternative kann man die Komponente als JavaScript-Objekt erstellen, um den Build-Schritt zu umgehen (siehe Listing 3.6). Diese 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. [16]

```
<script>
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 gibt es auch 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. [17]

```
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 weiter zugeben gibt es die *probs*-Option beim Erstellen einer Komponente. Das *probs*-Attribut erhält einen Namen, über die es dann später in HTML angesteuert werden kann. [16]

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

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

Listing 3.9: Erstellung eines *probs*

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

Listing 3.10: Nutzung eines *probs*

Abgesehen von den bereits aufgezählten Features, bietet Vue für Komponenten weitere Features wie die Möglichkeit auf Events zu reagieren, Inhalte mittels *<slot>* einer Komponente zu übergeben und die Verwendung dynamischer Komponenten. Eine Anleitung zur verwendung dieser Features ist in der Vue Dokumentation nachzulesen[3]. [16]

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

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. [15, S. 11]

Mustache-Syntax

Die einfachste Form der Datenbindung in Vue ist mit der auch aus anderen Frameworks bekannte Mustache-Syntax möglich. Bei der Mustache-Syntax wird ein Variablenname aus dem Modell der Komponente zwischen zwei geschweiften Klammern gesetzt. Angezeigt wird dann die aktuelle String-Repräsentation des Wertes. Die Mustache-Syntax ist unidirektionale vom Modell zur View. [18]

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

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

Listing 3.11: Mustache-Syntax

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 ein änderung im Modell. Ein Ändern in Teilen wie bei der Mustache-Syntax ist nicht möglich. [14]

```
<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 ist dafür da, wenn man Daten aus dem Model in der View als HTML interpretiert haben möchte. Aus Sicherheitsgründen empfiehlt es sich nur HTML-Code aus vertrauenswürdigen Quellen zu nutzen. [18]

```
<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

Möchte man Daten an HTML-Attribute binden, ist dies mit der *v-bind*-Direktive möglich. Da *v-bind* eine häufig genutzte Direktive ist, gibt es eine Kurzform, die einfach nur aus einem vorangestellten Doppelpunkt besteht. [18]

```
<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 updatet und bei änderungen im Model wird der Text im Inputfeld geupdatet. [19]

```
<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 Serve, die Seite wird dann serverseitig gerendert. Bei MPA handelt es sich um einen älteren *klassischen* Ansatz.[20, S. 262]
In Vue 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. [9, S. 174-175]

Installation

Der offizielle Router von Vue 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.[21]

Router initialisieren

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

```
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 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, wie z.B. eine Navigationsleiste. [?]

```
1  <div class="navbar">
2    <router-link to="/">Home</router-link>
3    <router-link to="/about">About</router-link>
4  <\div>
5  <router-view></router-view>
```

Listing 3.17: Verwendung des Routers

Routing mit Parametern

Möchte man in Vue Daten an eine aufgerufene Komponente übergeben, bietet Vue die Möglichkeit, dies via Parameter im Pfad zu machen. Um einen Parameter zu verwenden, muss man diesen beim Erstellen der Routen mit vorangestelltem : angeben. In der Komponente kann er dann wie in Listing 3.19 verwendet werden. [23]

```
1 const routes = [
2   { path: '/cars/:id', component: Cars },
3 ]
```

Listing 3.18: Route mit Parameter

```
1 $route.params.id
```

Listing 3.19: Zugriff auf Routingparameter

Weitere Routing-Features

Weitere Features die in Vue 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 sind ausführlich in der Dokumentation des Vue-Routers beschrieben.

3.6 State Management mit Vuex

3.7 Architektur

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

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] — *What is the HTML DOM?*, https://www.w3schools.com/whatis/whatis_htmldom.asp, (abgerufen 05.05.2023)

[3] — *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)

[4] — *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)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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