Hochschule
für Technik
Stuttgart

Bachelorarbeit

Konzeption und Realisierung einer Webanwendung zur Visualisierung von Positionsdaten

im Studiengang Wirtschaftsinformatik
der Fakultät Vermessung, Informatik und Mathematik
Sommersemester 2018

Name: Denise Müller
Matrikelnummer: 355097

Zeitraum: 03.04.2018 - 03.07.2018
Prüfer: Prof. Dr. Jan Seedorf
Zweitprüfer: M. Sc. Kevin Erath

Firma: IT-Designers GmbH
Betreuer: M. Sc. Kevin Erath

Abstract

Inhaltsverzeichnis

1 Einführung 1
- 1.1 Motivation . . . 1
- 1.2 Gesamtsystem . . . 1
- 1.3 Ziel der Arbeit . . . 1

2 Theoretische Grundlagen 2
- 2.1 VueJS . . . 2
  - 2.1.1 Framework . . . 2
  - 2.1.2 Funktionsweise/Anwendung . . . 4
  - 2.1.3 Reactive Programming . . . 6
  - 2.1.4 Andere Frameworks . . . 8
- 2.2 Architekturmuster Model-View-ViewModel . . . 10
  - 2.2.1 Motivation . . . 10
  - 2.2.2 Model-View-ViewModel . . . 11
  - 2.2.3 Verwandte Architekturmuster . . . 13
  - 2.2.4 Fazit . . . 15

3 Requirements 17
- 3.1 Funktionale Requirements . . . 17
  - 3.1.1 Funktionales Top Level Requirement . . . 17
  - 3.1.2 Requirements an die Anwendungsfälle . . . 17
- 3.2 Nicht funktionale Requirements . . . 18
  - 3.2.1 Anforderungen an den Nutzungskontext . . . 18
  - 3.2.2 Anforderungen an die Implementierung . . . 18

4 Anforderungsanalyse 19
- 4.1 Use-Case-Diagramm . . . 19

5 Systemarchitektur 20

6 Implementierung 21
- 6.1 Analysieren und Auswerten der Tracking Daten . . . 21
- 6.2 Darstellung Fahrbahn . . . 21
- 6.3 WebSockets Kommunikation . . . 21
- 6.4 Bluetooth Verbindung zu Modellautos . . . 21

7 Zusammenfassung/Ausblick 22

Abbildungsverzeichnis

2.1 Codebeispiel Initialisierung Vue 3
2.2 MVVM in VueJS 4
2.3 Komponente des MVVM Architekturmusters 11
2.4 Datenbindung zwischen View und ViewModel 12
2.5 Komponente des MVC Architekturmusters 14

4.1 Use Case Diagramm 19

Tabellenverzeichnis

2.1 Vorteile und Nachteile der Architekturmuster 16

Abkürzungsverzeichnis

MVVM Model-View-ViewModel

MVC Model-View-Controller

MVP Model-View-Presenter

HTML Hypertext Markup Language

CSS Cascading Style Sheets

DOM Document Object Model

CLI Call Level Interface

SPA Single-page Application

UI User Interface

1

1 Einführung

Die Gliederung des Hauptteils dieses Dokuments ist nur als Platzhalter gedacht. Im Folgenden sind zwei Vorschläge für Gliederungen dargestellt, die aber stets am konkreten Fall überprüft und in der Regel angepasst werden müssen.

Bitte beachten: Umfangreiche Programmlistings gehören nicht in die Arbeit, sondern auf eine beigefügte CDROM. Programmbeispiele können auszugsweise in der Arbeit gelistet werden, wenn sie zum Verständnis notwendig sind.

1.1 Motivation

1.2 Gesamtsystem

1.3 Ziel der Arbeit

2 Theoretische Grundlagen

Dieses Kapitel befasst sich mit dem Framework Vue.js und dem Entwurfsmuster Model-View-ViewModel (MVVM), das innerhalb der Arbeit genutzt wird und zum Verständnis der Arbeit dient.

2.1 VueJS

Die zunehmende Digitalisierung und die Nutzung sozialer Netzwerke bringen Sprachen und Technologien wie Hypertext Markup Language (HTML)[1]5 , Cascading Style Sheets (CSS)[2] oder JavaScript universell zum Einsatz. Egal welches Gerät gerade genutzt wird, ob Laptop, Computer oder Smartphone, jede Anwendung sollte auf allen Geräten lauffähig sein[1]. Für Anwendungen im Web wird ein Client als Browser und ein Webserver benötigt. Um die Daten, die durch HTMLangezeigt werden, zu manipulieren, interpretieren und zu aktualisieren, wird die Skriptsprache JavaScript benötigt[2]. Ebenso Anwenderinteraktionen wie Scrollen oder Klicken wird durch JavaScript interpretiert und verarbeitet.

2.1.1 Framework

Evan You ist der Gründer des JavaScript Frameworks **Vue.js**[3]. Vue.js ist ein clientseitiges Framework, mit dem Webanwendungen in JavaScript komponentenorientiert entwickelt werden können. Dies bedeutet, dass die Anwendung in kleine Teile unterteilt wird, die miteinander kommunizieren und als eigenständige Module laufen können. Das Framework setzt die Implementierung einer Webanwendung mit dem Architekturmuster Model-View-ViewModel durch und bietet hierdurch eine schlanke, universelle, anpassungsfähige und performante Implementierung[1]. Vue.js hat in den letzten Jahren immer mehr an Interesse gewonnen, was in der Google Trends Statistik deutlich zu erkennen ist [4].

Aufbau

Der Aufbau gleicht dem von Angular.js oder React.js: Frameworks, die im weiteren etwas genauer erläutert werden. Eine Anwendung besitzt mehrere Komponenten, die an sich einheitlich sind, sich aber von den anderen Komponenten trennen und über eine bestimmte Schnittstelle kommunizieren. Für die Benutzeroberfläche wird bekannterweise HTML benutzt, welche durch Attribute

1 Sprache, die es ermöglicht, Informationen im Internet zu präsentieren
2 Stilsprache, die das Aussehen von HTML-Dokumenten definiert

erweitert werden kann. Die Schnittstelle zur Kommunikation der Komponente wird an diese Attribute durch Datenbindung und Events realisiert. In diesem HTML Dokument stellt Vue.js viele Hilfsmittel für Styling und Formulare bereit. Um weitere Funktionen hinzuzufügen, werden Plugins in die Anwendung eingebaut[3]. Beispiele hierfür sind vue-router oder vue-custom-element. Um anzufangen, muss ein standard HTML Dokument erstellt werden, heißt es enthält ein `<head>`

```
<!DOCTYPE HTML>
...
 <BODY>
   <DIV ID="APP">
     <H1> {{MSG}}</H1>
   </DIV>
   <SCRIPT>
      NEW VUE({
         EL: "#APP",
         DATA() {
            RETURN {
              MSG: "HELLO WORD"
            }
         }
      });
   </SCRIPT>
</BODY>
```

Abb. 2.1: Codebeispiel Initialisierung Vue

und ein `<body>` Tag. Innerhalb des `body` Tags wird ein `div` mit einer Identifikation erstellt, beispielsweise wie in Abbildung 2.1 mit `id="app"`. Dort wird die Anwendung integriert. Um Daten wie einen String anzuzeigen, wird ein beliebiger Tag erstellt (zum Beispiel `h1`), der innerhalb einen Identifier besitzt, der in einer geschweiften Klammer (`<h1> {{msg}} </h1>`) steht. Um die `msg` mit einem String zu befüllen, wird ebenfalls innerhalb des `body` Tags ein `<script>` Tag definiert. Dort wird eine Instanz der Vue erstellt. Die Instanz besitzt ein Attribut namens `el`, das für die Identifikation zuständig ist, heißt hier sollte `#app` stehen. Des Weiteren hat die Vue Instanz eine `data()` Funktion, die Objekte, wie die `msg`, zurück gibt.

Model-View-ViewModel

Das Ausführen des Architekturmuster MVVM in Vue.js hat eine deutliche und klare Abgrenzung der Komponenten und kann durch das einfache Beispiel in Abbildung 2.2 veranschaulicht werden. Die Benutzerschnittstelle, die dem Anwender dargestellt wird, wird für gewöhnlich in HTML erfasst. In Vue.js wird dabei ein *div* mit einer Identifikation, im Beispiel der Abbildung 2.2 ist dies `app`, erstellt. In dem div soll ein Text stehen der durch das ViewModel übergeben werden soll. Dieser ist, wie auch in Angular oder ähnlichen Templating Frameworks, mit zwei

Abb. 2.2: MVVM in VueJS

geschweiften Klammern geschrieben. Der Text, der an diese Stelle eingefügt werden soll, wird im Model deklariert. Es wird ein Objekt erstellt, das ein Attribut `Text` enthält, was in diesem Beispiel ein String ist. Das Attribut muss genauso heißen wie das Wort innerhalb der geschweiften Klammer im HTML Dokument. Das Objekt wird an das ViewModel übergeben und dort in die Vue Instanz erstellt wird, übertragen. In der Vue Instanz muss die Identifikation des `div` stehen und das Objekt, das an die Oberfläche gegeben werden soll. Wenn das ViewModel die Instanz an das View schickt, erkennt die View die Identifikation und sucht in ihrem HTML Dokument nach der passenden Identifikation. Ist das `div` gefunden, werden die Daten durchsucht und sobald dann das Attribut `Text` gefunden wurde, wird dies an der Stelle mit dem gleichen Attributsnamen eingesetzt.

2.1.2 Funktionsweise/Anwendung

Routing

In Vue.js können Seiten durch Routing im HTML Dokument verlinkt werden. Die Seiten werden hierzu davor definiert. Das kann die Reaktionsfähigkeit der Webanwendung deutlich verbessern, da die Seite dynamisch auf dem Kontext der Seite aufgebaut ist, bedeutet, dass die Seiten einmal geladen werden und sonst nur angezeigt werden müssen und die einzelnen Bereiche aktualisiert werden, wenn sich beispielsweise Daten ändern. Dabei müssen die verschiedenen Views oder Seiten unterschieden werden. Vue.js unterstützt hierfür eine Router Bibliothek, **vue-router**[5]. Um die verschiedene Seiten zu definieren und zu verlinken, wird eine Instanz des Routers erstellt, in der ein oder mehrere Routen übergeben werden. Die Definition eines Routers ist ein Array, das mehrere Routen enthält, mit dem Attribut `path`, das die Verlinkung zur passenden Seite enthält. Diese Instanziierung wird an die Vue-Instanz übergeben. Um die Implementierung zu vollenden, muss im HTML Dokument die Routen aufgeführt werden um sie im Web wiederzugeben. Das wird wie folgt gemacht: `<router-view></router-view>`[6].

Templating

Templating ist eine Vordefinierung eines Designs oder eines Formats für ein Dokument. Dies kann für Vorlagen in Word oder ähnliches sein, aber auch für HTML Dokumenten als Vorlage für das Design der Webseite und Funktionen von einzelnen Komponenten. Diese Vordefinierung ist universell und kann mit jeden beliebigen Daten gefüllt werden[7]. Templating erlaubt, Werte bzw. Daten vom Model in die View zu binden. Seit Version 2.0 wird auch JavaScript Templating

mit HTML Templating in Vue.js unterstützt.
Das meist benutzte Symbol ist die doppelte geschweifte Klammer. Durch diese Art von Templating wird eine One-Way-Bindung vom Model zum Template aufgebaut. Mit einer One-Way-Verbindung können Daten von dem Model zum Template gesendet werden, aber nicht von dem Template zum Model[8]. Reaktive Datenbindung ist eine der Haupteigenschaften von Vue.js, sie speichert die Daten, wie Arrays oder JavaScript Variablen, in Verbindung mit dem HTML Dokument. Die One-Way Bindung, wie die Abbildung 2.2 zeigt, aktualisiert das HTML Dokument automatisch, wenn in JavaScript die `msg` manipuliert wird.
Um eine Two-Way Bindung zu erzeugen und über zum Beispiel einem Input Feld die Daten zu ändern, muss an das `input` Tag ein **v-model** integriert und an die Identifikation das zu ändernden Wertes gebunden werden. In Abbildung 2.1 müsste unterhalb der Zeile `<h1> {{msg}} </h1>` ein Input Feld mit der Bindung angefügt werden (`<input v-model="msg">`)[9]. Wird ein neuer Text in das Input Feld geschrieben, um somit den Wert zu ändern, wird der Wert durch die reaktive Datenbindung automatisch und mit einer geringen Reaktionszeit aktualisiert.

Events

Die Autoren Etzion und Niblett erklären in ihrem Buch „Event processing in action“ ein Event folgendermaßen:

> „ An event is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word event is also used to mean a programming entity that represents such an occurrence in a computing system[10].“

Das bedeutet, dass ein Event auftritt, wenn etwas passiert, wie beispielsweise ein Mausklick bzw. das drücken auf einen Bildschirms oder das Scrollen. Um ein Event in unser HTML Dokument an wie in etwa ein Button zum Klicken zu binden, wird in Vue.js ein `v-on` verwendet. Hierbei können auf verschiedene Art und Weise ein Button oder Methoden, die ausgeführt werden sollen, gebunden werden.
Für einen Button ist die `click`-Methode die standardgemäße Weise des Events. Für das folgende Beispiel wird ein Button zum hochzählen einer Zahl verwendet: `<button v-on:click="counter += 1">counter</button>`[11]. Das Klick-Event wird an den Button gebunden, sodass die Variable `counter` beim Eintreten des Events verändert wird.
Statt das Event an ein Objekt zu hängen, kann `v-on` auch an Methoden gebunden werden, um eventuell komplexere Ausführungen durchzuführen. Dabei kann ebenfalls eine `click`-Methode verwendet werden, die auf den Namen der Methode, die ausgeführt werden soll, hinweist (`v-on:click="Methodennamen"`)[11]. Zusätzlich können Parameter in dem Methodennamen angegeben werden, die in der Methode interpretiert werden können. Ebenso werden sogenannte *Modifier* von Vue.js unterstützt, um die immer wiederkehrenden Aufrufe während den Events handzuhaben. Modifiers sind Schlüsselwörter, die den Grad des Zugriffsrechts und die Sichtbarkeit auf Variablen, Funktionen oder Klassen regeln. Ein Beispiel wäre das `preventDefault`, das typischerweise aufgerufen wird, wenn das standardgemäse Verhalten des Browsers verhindert werden soll [11].

Validation

Eine Validierung durchzuführen, ist vor allem bei Formularen und Registrierungen wichtig. Die Validierung überprüft die Richtigkeit der Daten und die Erfüllung der gegebenen Anforderungen. Browser besitzten üblicherweise nativ die Validierung einer Form, da jedoch jeder Browser Objekte unterschiedlich handhaben können, ist die Vue.js basierte Validierung eine gute Lösung, um einheitlich zu bleiben. Zu Beginn sollte in dem HTML Dokument ein `form` Tag vorhanden sein, ?indem die Felder zur interaktiven Anwendung eines Formulars hinzugefügt werden?. Durch das schon bekannte `v-model` können Bedingungen an ein Attribut gebunden werden und in JavaScript anhand dessen verarbeitet werden können. Standardgemäß kann bei Eingabe einer Zahl, das über das `type` Attribut überprüft werden kann, ein Minimum und Maximum angegeben, das dann in Vue.js mit einem `v-if` überprüft wird und die passende Nachricht an den Anwender weiter gegeben werden kann. Diese Validierung erfolgt hauptsächlich in JavaScript, Clientseitig oder Serverseitig. Dabei gibt es Plugins, wie Vee-Validate und Vuelidate, die schon bestimmte Regeln mit sich bringen und das Validieren in Vue.js vereinfachen[12].
Mit Vee-Validate wird Clientseitig validiert und durch das Hinzufügen von einem `v-validate` Attribute, das auf das Model, das überprüft werden soll, weist. Vee-Validate bringt eigene Regeln mit, die mit `data-vv-rules` auf das jeweilige Model angepasst werden kann[**?**].
Vuelidate ist eine Modelbasierte Validierung, was eine flexiblere, auf das Minimum reduzierte Möglichkeit der Validierung ist. In dem von Vuelidate eigenen $v Schema werden die Validierungsmöglichkeiten gespeichert (`this.$v[propertyName]`). Das kann in dem HTML Dokument durch `v-if="!$v.emailValue.required"` auf Vorhandensein überprüft werden. Ebenso kann eine eigene Validierung erstellt werden, wenn die gewünschte Überprüfung nicht vorhanden ist, indem sie als Funktion dargestellt wird[**?**].

Komponente

Komponenten sind im Allgemeinen gesprochen Bereiche bzw. Teile eines Systems, die zusammen arbeiten können. In Vue.js helfen Komponenten, das standard-HTML Dokument zu erweitern. Um ein Komponent zu erstellen, muss dies erstmals mit `Vue.component(tag, constructor)` registriert werden, um die Komponente nutzen zu können[13]. In dem Konstruktor wird die Funktion definiert, die beim Verwenden des Komponenten ausgeführt wird, bzw. die Optionen mit den beinhaltenden Daten für das HTML Dokument. Eine Option, die vorhanden sein muss, ist die option `data`. `data` sollte für die Wiederverwendbarkeit eine Funktion sein, die das unabhängige Objekt zurück geben kann. Ist data keine Funktion, wird für jedes seperat erstellte Komponent das gleiche ausgeführt. Die Wiederverwendung kann per Name, der als `tag` übergeben wird, definiert werden. Heißt, wenn eine eigene Button-Komponente erstellt wird, kann man sie mehrmals verwenden, Beispiel die jedoch jeweils für sich ein eigener Button ist und die Funktion, die hinter der Komponente steht, wird für jeden Button seperat ausgeführt, da die Instanz jedes mal aufs neue erstellt wird[13].

2.1.3 Reactive Programming

Reactive Programmierung wurde in den letzten Jahren immer mehr zum Trend[14]. Reactive Programmierung ist ein Programmierstil, der vor allem in event gesteuerten und interaktive An-

wendungen gut genutzt wird. Bekannte Konzerne wie Amazon[1] und Netflix[2] benutzen bereits reaktive Programmierung. Dabei werden Zustandsänderungen für die bekannt gemacht, die sich dafür interessieren. Wenn zum Beispiel eine Funktion aufgerufen wird, die zwei Zahlen addiert, würde in dem üblichem Programmierstil (*imperativ*) die Funktion die Variable der Summe ändern. Wenn nach der Ausführung der Funktion die Variablen sich ändern würden, hätte das keinerlei Auswirkungen auf die Summenvariable. Bei einem reaktiven Programmierstil nimmt die Summenvariable den aktuellen Wert der beiden addierten Variablen an[15].
Reactive Programming ist ein Teil aus Objektorientierter und ein Teil aus Funktikonaler Programmierung, das asynchrone und immutable Streams von Events beinhaltet. Die Streams werden miteinander kombiniert und werden von `Observables` „abgehört“. Jedoch muss bei den `Observable` zwischen `Hot` und `Cold Observables` unterschieden werden. Cold `Observables` sind mit Listen vergleichbar, da wie gewohnt bei der Erzeugung die Größe fest steht und der Inhalt kann dementsprechend nicht verändert werden. Hot `Observables` sind das genaue Gegenteil, hierbei ist nicht bekannt, wie der Inhalt aussehen könnte oder wie groß das Objekt wird. Hierbei wird ein Zeitintervall erstellt, dass innerhalb der angegebenen Zeit ein Event sendet, an das sich das Objekt abbonieren (engl. *subscriben*) und somit Veränderungen der Daten registrieren kann.[16]. Diese ganze Verarbeitung läuft in der reaktiven Programmierung asynchon ab und macht die Programme effizienter, da bei einer asynchronen Verarbeitung im Gegensatz zur synchronen Verarbeitung nicht gewartet wird, bis das Ergebnis einer Methode zurück gegeben wird, sondern läuft im Code weiter und gibt das Ergebnis später, sobald es vorhanden ist zurück.

Reaktives System

Um die Anforderungen, Ziele und den Aufbau eines Reaktives System zu definieren, wird ein Manifest geschrieben. Für das Reactive Programming heißt das Manifest „*Reactive Manifesto*“[17]. Oftmals werden reaktive Systeme als schneller und robuster bezeichnet, dass sich mit den vier Eigenschaften beschreiben lässt:

- **Responsive:** Die Antwortbereitschaft des Systems und die Erkennung von Bugs ist unmittelbar[16]. Die Fehler können jedoch nur durch Abwesenheit einer Antwort garantiert erkannt werden, außerdem sollten hierfür die Zeit bis zur erwarteten Antwort eingegeben werden[17].
- **Resilient:** Die Widerstandsfähigkeit des Systems wird bewiesen durch die immer noch vorhandene Antwortbereitschaft während eines Systemfehlers oder von Ausfällen von Hard- oder Software. Implementiert werden kann dies durch Isolation von Komponenten, Eindämmung von Fehlern, Replikation von Funktionalitätem und das Delegieren der Verantwortungen[17]. Anfragen warten in einer bestimmten Zeit auf eine Antwort, wenn diese nicht kommt, weiß der Service, der die Anfrage gestellt hat, bescheid und kann ihn zeitnah behandeln[16].
- **Elastic:** Elastisch bedeutet, dass das System auch durch verschiedene Lastbedingungen einen konstanten Service liefert. Für die Erkennung der Veränderungen und das darauf reagiert werden kann, muss auch hier die Replikation von Funktionalitäten gegeben sein. Bei erhöhter Last werden die Replizierungsfaktoren darauf angepasst, dabei sollten keine

1 https://www.amazon.de/p/feature/j5d6uh4r8uhg8ep
2 https://help.netflix.com/de/node/68708

Einschränkungen vorhanden sein[17]. Cloud-Dienste sind hier eine gute Lösung, um die Problematik einzugrenzen und das System kosteneffektiv zu betrieben[16].

- **Message Driven:** Nachrichtenorientierte Systeme benutzen eine asynchrone Nachrichtenkommunikation. Dabei werden die Komponente, die miteinander kommunizieren, entkoppelt und Isoliert. Dabei können Fehler an andere eventuell übergeordnete Komponente gesendet werden. Die Systeme lassen sich an die Nachrichten hängen, was die Überwachung von Veränderungen verinfacht[16]. Die Nachrichtenüberwachung veranlassen einen Überblick über das Laufzeitverhalten des Systems und der übermittelten Nachrichtenflüsse. Bei Nachrichtenorientierte Systeme ist das Programm auch ortsunabhängig möglich, bedeutet, dass Teile des Codes nicht auf demselben Computer ausgeführt werden müssen[17].

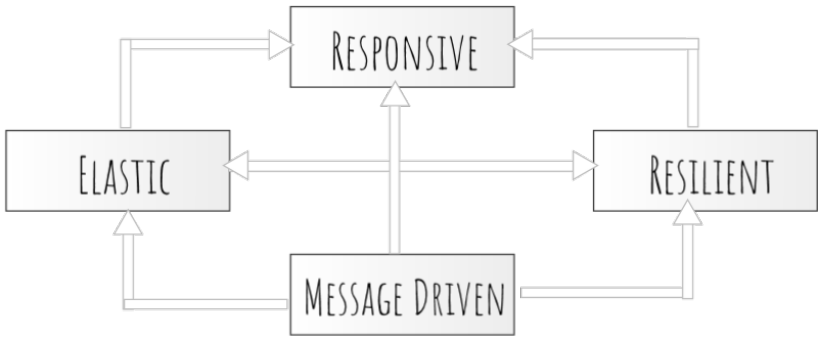

Abb. 2.3: Zusammenspiel reaktives System

Auf Abbildung **??** wird das Zusammenspiel der vier reaktiven Qualitäten illustriert. In großeren Anwendungen gibt es mehrere Komponenten, die voneinander abhängig sind. Deshalb müssen die vier Qualitäten in jeder Ebene des Gesamtsystems berücksichtigt werden und es somit innerhalb der Schichten kombinierbar[17]. Zur Umsetzung eines reaktiven Programms gibt es Frameworks, die die Anforderungen, Eigenschaften und Ziele eines reaktiven Programmes helfen umzusetzen. Beispiele hierfür wären Vue.js, React.js und weitere.

2.1.4 Andere Frameworks

Analog zu Vue.js gibt es weitere Bibliotheken bzw. Frameworks, die die Anforderungen ebenfalls als Reactive Programming betreiben und mit dem Entwurfsmuster MVVM oder Model-View-Controller (MVC) arbeiten. Ein Beispiel wäre **React.js**. React.js ist eine von Facebook erstellte JavaScript Bibliothek, das dem Entwickler beim Erstellen von Oberflächen hilft. Konzerne wie Whatsapp, Instagram und Facebook nutzen die Frontend Bibliothek. Das Ziel, das React.js verfolgt, ist es einfacheren Code zu schreiben, um die einzelnen Bestandteile besser zu verstehen und weniger komplex zu halten. Die wesentliche Bestandteile von React.js sind die Komponentenarchitektur, der virtuelle Document Object Model (DOM) und die Browserkompatibilität.
React.js Komponente sind äquivalent zu Web Komponenten, die mit `React.createClass` erstellt werden. Innerhalb gibt es Funktionen wie `render`, die das HTML Dokument für das Web präsentierbar macht und bei jeder Änderung aufgerufen wird. Dazu können eigene Funktionen definiert werden. Bei React.js wird die standard `onClick` Methode als Attribut auf den Button gesetzt und durch geschweifte Klammer kann auf die Funktionen verwiesen werden (`<button`

`onClick=this.add>`). Ein besonderes Attribut von React.js ist das `State`. Dieses Attribut beinhaltet die zu verändernden Daten und kann die Aktualisierung im HTML Dokument durchfüren. Somit muss auf die Anpassung des DOM keine Rücksicht genommen werden. Die Komponente werden in React.js innerhalb der `render` Methode definiert, da React.js JSX eine schlanke Syntaxerweiterung zum Schreiben von Markups verwendet. Änderungen am Stil der Seite oder des Buttons wird innerhalb der `createClass`, wo auch die `render` Methode für die Definition des HTML Dokuments implementiert wird, erstellt (`return backgroundColor: #fff;`). Hierbei wird erkannt, dass die üblich bekannte Trennung der Bereiche HTML, CSS und JavaScript nicht stattfindet[18].
Die Bearbeitung durch JSX in React.js wird in der Regel nicht direkt in dem DOM des Browsers stattfinden, sondern mit einem virtuellen DOM, das ein JavaScript Objekt ist, das zur Bearbeitung genutzt wird. Bei einer Veränderung wird jeweils ein neues Objekt, also ein neuer virtueller DOM, erstellt. Dabei wird der virtuelle DOM mit dem Browser DOM verglichen und aufgelistet. Die Änderungen werden erst zum Zeitpunkt des Batch an den Browser geschickt und aktualisiert[19]. Im Vergleich zu Vue.js ist React.js einer der ähnlichsten Bibliotheken. Beiden nutzen einen virtuellen DOM, eine reaktive, zusammensetzbare Benutzeroberfläche und nutzen durch das Routing und des States um die Änderungen zu fokussieren. Die folgende Auflistung zeigt die Unterschiede in Vue.js und React.js in den Positionen der Leistung, im Templateing und JSX und die Skalierbarkeit des Systems.

- **Leistung:** Vue.js und React.js sind in Hinsicher der Leistung ähnlich schnell, was für die Entscheidung irrelevant ist[20].
- **Templating** Im Gegensatz zu Vue.js ist in React.js HTML und CSS zusammen in JavaScript mit Hilfe von JSX geschrieben, das mit Hilfe einer `render` Funktion, die ebenfalls in Vue.js vorhanden ist, an der Benutzeroberfläche daregestellt werden kann. Ebenso werden Werkzeuge wie Typenüberprüfung in React.js besser unterstützt[20]. Die konkrete Teilung zwischen JavaScript, HTML und CSS ist in Vue.js deutlich erkennbar und für die Meisten Entwickler übersichtlicher[21]. In Vue.js ist das Templating für das Rendering des HTML Dokuments. Für viele Entwickler, die in HTML Erfahrung haben, wird es einfacher sein, den Code zu lesen und zu interpretieren[20].
- **Skalierbarkeit:** Für die Skalierung für größere Anwendungen gibt es in Vue.js sowie in React.js viele Routing Lösungen. In React.js gibt es die Frameworks Fluex oder Redux und in Vue.js das Vuex, auch Redux kann in Vue.js integriert werden[22]. Vue.js beinhaltet einen Call Level Interface (CLI), das die Einbindung neuer Projekte mit Werkzeugen zum Bauen des Projekts wie webpack oder Browserify ermöglicht. Vor allem in React.js müssen die Build Systeme, zum Bauen des Projekts sich zuerst angeeignet werden, was in Vue.js beispielsweise das Webpack übernimmt[20].

Ein weiteres Framework, das für die Frontend Entwicklung hilfreich ist, ist **Angular.js**. Angular.js ist den Meisten bekannter als Vue.js oder React.js. Viele Komponenten oder Syntaxen von Angular.js waren für Vue.js Inspirationen, weshalb es vielen Angular.js Entwicklern einfacher fällt sich in Vue.js einzuarbeiten. Angular.js ist deutlich komplexer und hat mehr Vorgabe in Hinsicht der Softwarearchitektur als Vue.js. Ebenso ist die Leistung von Vue.js deutlich besser und leichter zu optimieren.
Die Entscheidung für Vue.js basierte auf der immer größerer werdenden Popularität seit 2016, was durch die Google Trends Statistik deutlich hervorgeht[4]. Für die Größe des zu implementierende Projekt ist Angular.js zu überladen. Vue.js oder React ist kompakt genug, für eine

Webanwendung zur Darstellung von Positionsdaten jedoch ausreichend. Dabei ist die Leistung sehr entscheidend.
in einer Studie einer Bachelorarbeit aus Schweden wurde der Performancevergleich von Angular 2, Aurelia, Ember, Vue und weiteren getestet. Dabei wurden die Befehle `Create`, `Delete` und `Update` mit jeweils 1000 Reihen getestet.[23]

- **Angular.js 2:** Im Ganzen hatte Angular 2 in jedem Testszenario die beste Leistung, dabei hat Angular 2 eine deutliche Verbesserung zu seinem Vorgänger 1.5[23].
- **Aurelia:** Aurelia hat in dem Szenario das zweitbeste Ergebnis erziehlt, jedoch schnitt Aurelia bei dem Befehl Update mit unter Anderem als schlechtestestes Frameworks ab[23].
- **Ember:** Eins der schlechtesten Frameworks ist Ember. Nur in dem Befehl Delete erreichte Ember einen durchschnittliches Ergebnis[23].
- **Vue.js:** Vue.js war eins der schnellsten Frameworks, vor allem bei den Befehlen Create und Update[23].

2.2 Architekturmuster Model-View-ViewModel

2.2.1 Motivation

Naveen Pete, ein Online-Blogger, bezeichnete ein Architekturmuster, auch Entwurfsmuster genannt, als ein „gut strukturiertes Dokument“, dass als Lösung für wiederkehrende Probleme dienen soll[24].

> „A design pattern is a well-documented solution to a recurring problem.“

Architekturmuster werden zur sauberen Trennung der Anwendungslogik und der Benutzeroberfläche genutzt. Das vereinfacht das Testen, die Instandhaltung und das Weiterentwickeln der Software. Außerdem verbessern Architekturmuster die Wiedernutzung des Codes für Andere Projekte, da lediglich der View Part plattformspezifisch angepasst werden muss[25]. Architekturmuster werden vor allem in großen Projekten genutzt, um eine Abgrenzung zwischen „Ansicht“ und „Modell“ zu schaffen. In dem Architekturmuster MVVM ist eine klare Abgrenzung zwischen der grafischen Oberfläche (View) und die Datenhaltung (Model) durch eine Schnittstelle (ViewModel) gegeben.

Model

In den üblichen Architekturmustern wird das Model als „Abbildung der Datenquelle“ gesehen. Für das Architekturmodell MVVM ist das Model eine Abbildung der Daten für die Visualisierung. Diese Daten werden vom Benutzer manipuliert und zur Verfügung gestellt. Das Model stellt dabei folgende Funktionalitäten bereit:

- Validierung
- Benachrichtigungen bei Änderungen
- Verarbeiten nach vorgegebenen Regeln

Was dabei zum Einsatz kommt, hängt von den Anforderungen an das Model ab. Werden die Daten beispielsweise von einem OR-Mapper oder einem Service zurück geschickt, könnten diese Funktionalitäten ebenso in das ViewModel implementiert werden[26].

View

Die View ist die strukturierte Benutzeroberfläche, in der Daten, Videos sowie Bilder dargestellt werden und keinerlei Logik enthalten ist. Benutzereingaben, wie über die Tastatur werden in der View abgefangen und an das ViewModel weiter gegeben. Die View ist ausschließlich mit dem ViewModel verbunden und das nur, wenn die Daten an die View überreicht werden. Im Code sollte in der View so wenig wie möglich geschrieben und unabhängig sein. Das heißt, der Code sollte sich ausschließlich auf die View beziehen. Dies ermöglicht das einfache Austauschen der View, ohne große Änderungen am Code zu leisten[27].

ViewModel

Einfach gesagt, stellt das ViewModel das Model für die View dar und gibt das Model nach außen, heißt das ViewModel bearbeitet die Logik der View. Das ViewModel kommuniziert mit dem Model durch Methodenaufrufe und stellt die Daten, das das ViewModel vom Model geliefert bekommt, der View dar. Die zur Verfügung stehenden Funktionalitäten werden durch die View gebunden, wodurch für die View keinerlei Code anfällt. Referenzen auf Elemente der View dürfen nicht erstellt und darauf zugegriffen werden, da, wie in dem Abschnitt der View bereits erwähnt wurde, keine Abhängigkeiten erstellt werden dürfen. Durch Funktionen über die View zu testen ist unpraktisch und entfällt hier, da das ViewModel selbst die Abstraktion der View die Möglichkeit besitzt, das abzudecken. Das ViewModel bezieht sich niemals auf die View und kann somit auf jede View bezogen werden und macht sich dafür wiederverwendbar[28].

2.2.2 Model-View-ViewModel

Durch die Entwicklung des Windows Presentation Foundation Framework (WPF) durch Microsoft[1] entstand 2005 das Architekturmodell MVVM, das ein fester Bestandteil von Silverlight[2] ist. Ebenso werden existierende Frameworks um MVVM erweitert[29].

Wie die drei Komponente View, ViewModel und Models miteinander zusammenhängen und kommunizieren, wird im folgenden erläutert und durch Abbildung 2.3 abgebildet.

1 https://www.microsoft.com/de-de

2 Silverlight is a cross-browser, cross-platform plug-in for delivering media and rich interactive applications for the Web

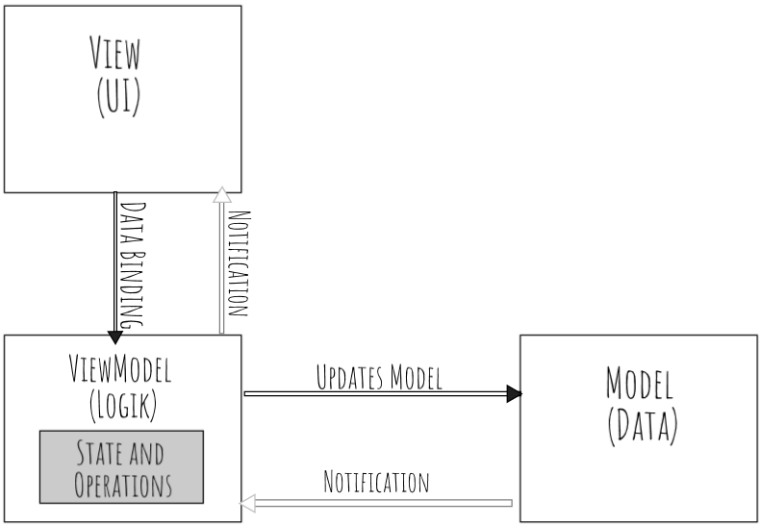

Abb. 2.4: Komponente des MVVM Architekturmusters

Data-Binding

Wie schon davor erwähnt wurde, kennt die View das ViewModel und das ViewModel kennt das Model, jedoch nicht vice versa, was die Pfeile in Abbildung 2.3 illustriert. Damit die View mit dem ViewModel interagieren kann, wird in WPF ein sogenanntes Data-Binding verwendet. Durch die Datenbindung wird eine Verbindung zwischen der View (User Interface (UI), die Benutzerschnittstelle) und dem ViewModel (Logik) hergestellt. Wenn das Model (Daten) die korrekten Benachrichtigung bereitstellt, können die eingebundenen Daten automatisch die Änderung des Wertes annehmen und in der View wiedergeben[30]. Eine Verbindung besteht zumeist aus vier Komponenten: ein Zielobjekt, eine Zieleigenschaft, die eine Abhängigkeitseigenschaft sein muss, eine Quelle und ein Pfad zum Wert, das verwendet wird. Wenn eine Verbindung aufgebaut ist, ist die Richtung des Datenflusses entscheidend und wichtig. Hier gibt es drei verschiedene Wege die Daten zu senden bzw. zu empfangen.

- One-Way Bindung
- Two-Way Bindung
- One-Way To Source Bindung

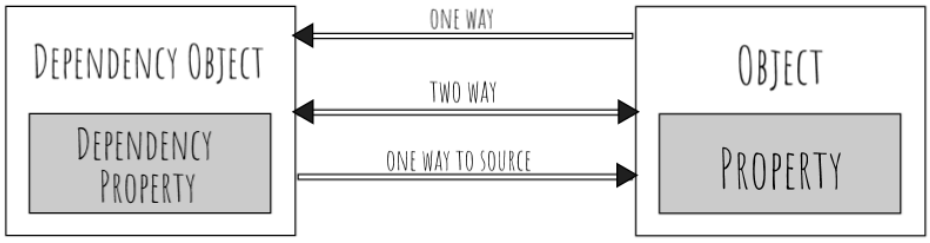

Abb. 2.5: Datenbindung zwischen View und ViewModel

Die View, das Bindungsziel, besitzt ein Dependency(engl. für abhängig) Objekt mit einer Dependency Property, einer Eigenschaft, wie zum Beispiel einem String. Diese sind abhängig voneinander. Das ViewModel, die Bindungsquelle, besitzt ein Objekt mit einer Property. Eine One Way Bindung lässt Änderungen an der Quelle zu und aktualisiert die Zieleigenschaften automatisch. Diese Verbindung ist vor allem nützlich, wenn schreibgeschützte Elemente in der UI vorhanden sind, denn Änderungen an den Zieleigenschaften sind bei einer One Way Bindung nicht möglich. Eine Two Way Bindung ermöglicht dem User, Änderungen vorzunehmen. Hierbei werden die Zieleigenschaften verändert und automatisch die Quelleigenschaften aktualisiert. Andersherum geschieht genau das gleiche, analog zur One Way Bindung. Für zum Beispiel zu bearbeitende Formulare oder andere interaktive Situatuionen, die vermutlich Abhängigkeitseigenschaften aufweisen, ist diese Bindung rätlich. Die konträre Bindung zur One Way Bindung ist die One Way To Source Bidnung, diese aktualisiert automatisch die Quelleigenschaften wenn die Zieleigenschaft geändert wurde.

Notifications

Um die Änderung zu erkennen, muss die Quelle einen Benachrichtgungsmechanismus beinhalten. In der Regel ist dies die `INotifyPropertyChanged` Implementierung. Bei dieser Implementierung wird ein Objekt an die UI gebunden, das das Ereignis `PropertyChanged` enthält und dies auslöst, sobald eine Änderungen an der Eigenschaft des Objektes wahrgenommen wird. Diese Änderung wird von dem Model an das ViewModel gesendet, das dann die Aktualisierung an der Quelleigenschaft vornehmen kann und die Änderung in der View angezeigt. Die Quellaktualisierungen werden anhand `UpdateSourceTrigger`, die Eigenschaften enthalten, die bestimmen, wann und warum die Benachrichtigung ausgelöst wird, ausgelöst.

Fazit

Das Ziel des Architekturmuster ist, wie auch bei dem Architekturmuster MVC, das der Vorreiter von MVVM ist, das Trennen der Logik und der Präsentations Schicht bzw. der View. Die Vorteile des Entwurfsmuster MVVM sind:

- Durch die Trennung müssen Änderungen, die an dem Model vorgenommen werden, nicht an der View geändert werden[31] .
- Während der Entwicklung können die Entwickler und die Designer unabhängig voneinander an den Komponenten arbeiten[24].
- Modultests für das ViewModel und das Model können ohne die View erstellt werden[24].
- Die View kann beliebig ausgetauscht und wiederbenutzt werden, da die View unabhängig von den anderen Komponenten ist[24].
- Die Weiterentwicklung sowie die Instandhaltung kann durch die Trennung detaillierter vorgenommen werden, ohne dass die Änderungen negative Auswirkungen auf das System haben[31].

Die Nachteile des Architekturmuster sind in der Minderheit. Für viele Entwickler ist MVVM für einfache Oberflächen zu mächtig und ebenso für größere Fälle kann es schwierig sein, das ViewModel zu konstruieren. Wenn die Datenbindung komplexer ist, wird das Debugging ebenso problematischer[31].

2.2.3 Verwandte Architekturmuster

2.2.3.1 Model-View Controller

Das Hauptmerkmal des Architekturmusters MVC ist die Trennung der View und des Controllers. Genau das macht das MVC komfortabel für Web Anwendungen und weniger geeignet für Desktop Anwendungen[32]. Das Model ist beim Architekturmuster MVC der Ort, an dem die Daten und Objekte sowie der Netzwerkcode gespeichert und implementiert werden. Das heißt, der Model Part beinhaltet die Informationen, um die View anschaulich zu machen und die Logik, die die Änderungen des Users verarbeitet und die View damit benachrichtigt[33].
In der View ist ebenso wie in dem MVVM Architekturmuster keinerlei Logik enthalten, was es wiederverwendbar für andere Projekte macht.[34] Die View setzt Informationen für den Anwender ins Bild. Mit dem Controller zusammen definieren sie das UI (Benutzerschnittstelle)[33].
Der Controller vermittelt vorwiegend über „delegation pattern“ zwischen der View und dem Model. Delegation Pattern ist eine Technik, bei der Objekte Verhalten nach außen darlegen und das Verhaltens an ein anderes Objekt übertragen[35]. Idealerweiße kennt der Controller die View nicht und kommuniziert über ein bestimmtes abstraktes Protokoll[34]. Der Controller verarbeitet die Events der View und leitet sie an das Model weiter, das dann die View über die Änderung benachrichtigen kann und die View die Änderung dem Anwender darstellt. Vergleichbar mit dem ActionListener in Swing[36].

Model-View-Controller

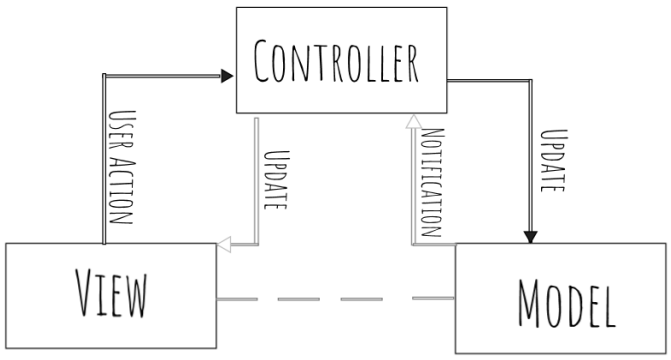

Abb. 2.6: Komponente des MVC Architekturmusters

Auf Abbildung 2.5 erkennt man die Zusammenhänge zwischen den Komponenten des Architekturmodells MVC. Dieses Modell lässt mehrere Views und Controller zu, die unabhängig von dem Model erstellt und modifiziert werden können[37]. Wenn der Anwender eine Aktion in der View auslöst, wird diese an den Controller geschickt, der anhand der Manipulation das Model aktualisiert. Sobald die Aktualisierung abgeschlossen ist, wird eine Benachrichtigung an den Controller

gesendet, ob diese erfolgreich war oder nicht. War die Aktualisierung erfolgreich, aktualisiert der Controller anhand der Benachrichtigung die View.

Kommunikation

Obwohl das Model unabhängig von den Views ist, sollten die Komponente miteinander kommunizieren. Ebenso die Controller und die Views. Da die View jedoch das Model kennt, kann die View mit dem Model in Kontakt treten. Die Kommunikation mit den Nachrichten findet über **Events** (Aktionen) statt. Events stellen Mechanismen bereit, die mit wenigen Abhängigkeiten eine Kommunikation zustande kommen kann. Grafische Komponente, wie Listen, Textfelder oder ähnliche können Benachrichtigungen empfangen, beispielsweise wenn geklickt wird. Diese Benachrichtigungen kommen in der Regel vom Controller. Die View registrieren die Events an Objekte, die sie bearbeiten möchten. Wenn die Nachricht zum Model gesendet wurde, wird das Application Model auf diese Nachricht antworten, die Daten aktualisieren und eine Nachricht zurück senden. Das Application Model kann ebenso events auslösen und somit die abhängigen Views überwachen[38].

2.2.3.2 Model-View Presenter

Das Architekturmuster Model-View-Presenter (MVP) basiert auf dem MVC Muster. Die Komponente wurden anhand hohen Flexibilität angepasst und die Mängel des vorherigen Musters (MVC) werden besser gehandhabt. Das Hauptmerkmal des MVP Musters ist der Presenter, der direkten Zugriff auf die View und das Model besitzt und deren Zusammenspiel handhabt. Die View und das Model können kleine Situationen selbst meistern, was das ganze die Komplexität des Presenters reduziert, da dieser nur noch für die komplexen Anforderungen verantwortlich ist. Somit ist das Architektumuster MVP das flexibelste der MV* Familie, da es dem Entwickler viel Freiraum und Kontrolle gibt. Zum Beispiel kann die Datenbindung da genutzt werden, wo es dem Entwickler am besten passt[32]. Die Views sind wie auch beim MVC Muster für die Anschauung der Daten zuständig und das Model für die Datenhaltung verantwortlich. Martin Fowler verglich 2006 das Architekturmuster MVC mit MVP so, dass der Controller in MVC bei MVP ein Teil der View ist[39]. Die Reaktion auf die Aktivitäten des Users ist im Presenter enthalten. Er kann entscheiden wie das Model manipuliert und verändert werden kann, heißt er übernimmt bzw. integriert die Rolle des Application Model, das man von MVC kennt. Zusammenfassend zu sagen ist, dass der Presenter die Business Logik zur Verfügung stellt. In der Regel verhält sich die Kommunikation gleich wie beim MVC. Durch Aktionen des Anwenders, wie eine Mausbewegung, werden `Interactor Events` ausgelöst. **?** Diese Interaktionen werden vom Presenter interpretiert[40].

2.2.4 Fazit

Bevor Software-Projekte mit Architekturmuster realisiert worden waren, wurde die grafische Oberfläche in die Mitte der Anwendung implementiert. Dieser einfache Einsatz wird Smart UI anti-pattern genannt, das wie jedes Muster seine Vorteile hat, aber bei größeren Projekten zu einer Hürde werden kann. Das Gestalten der verschiedenen Architekturmustern bewältigen die Probleme des Smart UI anti-pattern. Jedes Architketurmuster ist ähnlich aufgebaut, jedes trennt

die Benutzerschnittstelle mit den Daten und deren Verarbeitung. Die Kommunikation zwischen den drei Komponenten hängt von der grundlegenden Umgebung ab. Beispeilsweise ist die Synchonisation mehrerer Views durch einen „observer", einen Beobachter, der die Änderungen weitergibt und somit die View aktualisiert, nützlich, aber in vielen Situationen nicht zugänglich. Somit sind auch andere Mechanismen brauchbar, wie die Datenbindung die in dem Architekturmuster MVVM zwischen dem ViewModel und der View. Die Anwendung des Mechanismus zur Kommunikation hängt von der Anwendung selbst, der Programmiersprache, der Frameworks, des angewendeten Musters und der persönlichen Preferenz aufgezeigt[39]. In der folgenden Tabelle[32] werden die Vorteile und Nachteile des jeweiligen Entwurfsmusters. Dies zeigt, dass kein Gewinner ausgewählt werden kann, jeder hat seine eigenen individuellen Vorzüge.

Tab. 2.1: Vorteile und Nachteile der Architekturmuster

Muster	Vorteile	Nachteile
MVVM	Unterstützt mehrere Views für das gleiche Model und aktualisiert View und ViewModel automatisch	Das Benutzten von „Beobachter" schwächt die Leistung. Beruht auf der zugrunde liegenden Technologie.
MVC	Gut geeignet für Web Anwendungen, da View und Controller hier getrennt gehandhabt wird.	Die Abhängigkeit von View und Controller, was hier nicht gegeben ist, sind in manchen Steuerelementen von nöten.
MVP	Flexibilität und Freiheit zum Verwenden der verschiedenen Mechanismen und kann an viele Anwendungsszenarien verwendet werden.	Keine strikte Trennung der Komponenten, das bei komplexere Codestellen zu Problemen führen kann.

3 Requirements

Requirements, in deutsch „Anforderungen“, beschreiben die Ziele eines Kunden, die angeforderten Funktionalitäten und Eigenschaften des Systems, sowie in welcher Qualität dies umgesetzt werden muss[41]. Die Beschreibung, das Prüfen, das Verwalten sowie die Analyse solcher Requirements wird als des Requirements Engineering beschrieben[42].

3.1 Funktionale Requirements

Funktionale Requirements charakterisieren die Leistungen an das System von Anwendungsfällen. Zusätzlich definieren die funktionalen Requirements die technischen Funktionen eines Systems, die als Lösung für die Problematik der Anwendungsfälle fungieren[43].

3.1.1 Funktionales Top Level Requirement

Requirement 1000:
Das System soll Anwendern eine Fahrbahn visualisieren und die darauf vorhandene Position eines Fahrzeugs aufzeigen.

3.1.2 Requirements an die Anwendungsfälle

- **Requirement 1110:**
  Der Anwender kann das Fahrzeug vom System aus ein- und ausschalten.
- **Requirement 1120:**
  Anhand von Positionsdaten soll die Fahrbahn im Web visualisiert werden.
- **Requirement 1130:**
  Anhand von Positionsdaten soll ein Fahrzeug (optional mehrere Fahrzeuge) im Web angezeigt werden.
- **Requirement 1140:**
  Optional: Der Anwender kann durch Eingabe einer Zahl die Geschwindigkeit des Fahrzeugs anpassen.

3.2 Nicht funktionale Requirements

Nicht funktionale Requirements definieren Forderungen an den Lösungsbereich wie die Architektur, Technologien oder an die Qualität wie zum Beispiel die Performance. Beispiele hierfür sind sie Bedienbarkeit, das eingesetzte Betriebssystem, die verwendete Hardware und die genutzte Programmiersprache[43].

3.2.1 Anforderungen an den Nutzungskontext

- **Requirement 1210:**
  Das System soll über eine Webanwendung verfügbar sein.

Darstellung:

- **Requirement 1220:**
  Durch Intervalländerungen der Positionen, sollen die Änderungen zeitlich schnell aktualisiert werden.
- **Requirement 1230:**
  OPTIONAL: Die Darstellung des Fahrzeugs in der Webanwendung sollte in einer reibungslose Bewegung dargestellt werden

3.2.2 Anforderungen an die Implementierung

Architektur:

- **Requirement 1310:**
  Die Anwendung zeigt eine Single-page Application (SPA)-Architektur auf. SICHER?

Technologien:

- **Requirement 1320:**
  Das System soll mit der Frontend Bibliothek Vue.js umgesetzt werden.
- **Requirement 1330:**
  Die Positionsdaten sollen per Webservices, wie Websockets übermittelt werden.
- **Requirement 1340:**
  Die Verbindung zu den Fahrzeugen soll durch Web-Bluetooth verbunden werden.
- **Requirement 1350:**
  OPTIONAL: Durch die gegebene Zeit und der Strecke soll die Geschwindigkeit durch Berechnungen dargestellt. werden.

4 Anforderungsanalyse

Erklärung Anforderungsanalyse

4.1 Use-Case-Diagramm

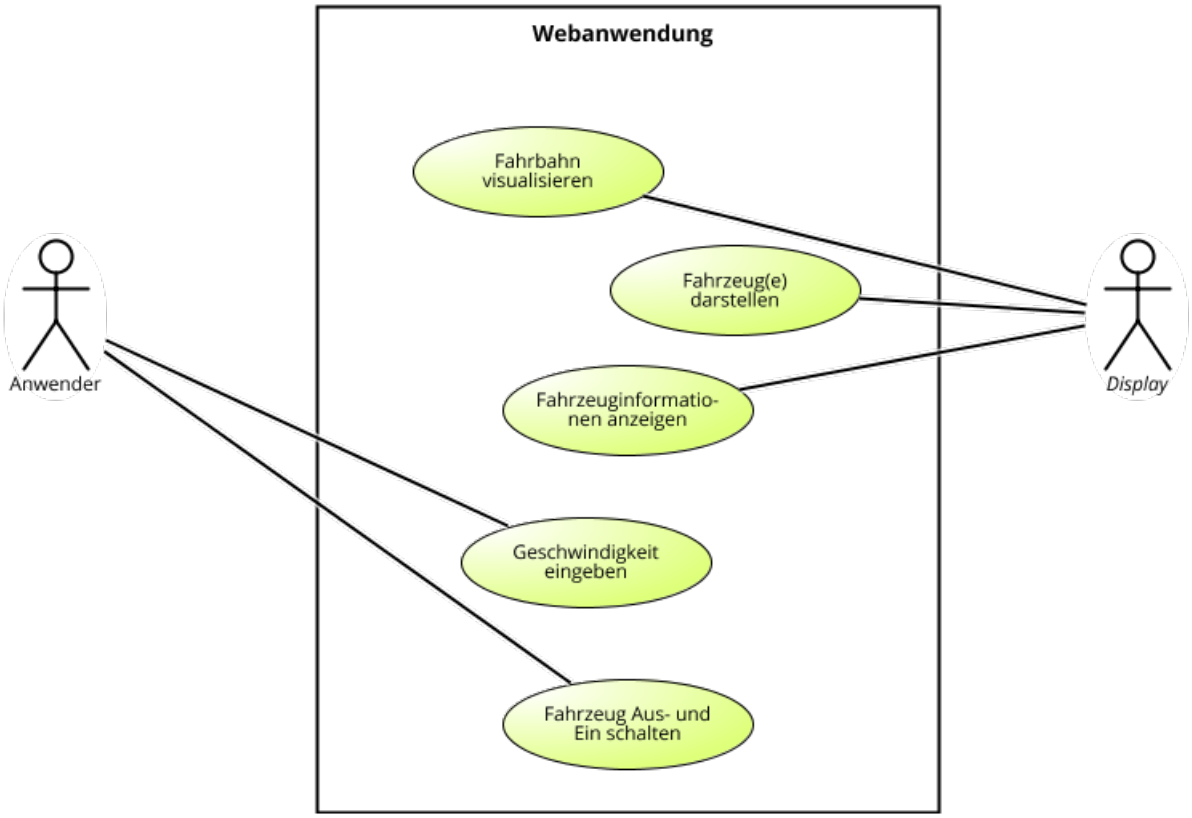

Abb. 4.1: Use Case Diagramm

5 Systemarchitektur

6 Implementierung

6.1 Analysieren und Auswerten der Tracking Daten

1. auswerten

6.2 Darstellung Fahrbahn

1. fahrbahn darstellen

6.3 WebSockets Kommunikation

1. webSockets

6.4 Bluetooth Verbindung zu Modellautos

1. bluetooth

7 Zusammenfassung/Ausblick

Ergebnis-Bewertung, Zusammenfassung und Ausblick

Literaturverzeichnis

[1] Marc Teufel. Vue.js: Der siegeszug eines frameworks. *entwicklerspezial, entwickler.de*, 2018. URL https://entwickler.de/leseproben/vue-framework-579832700.html.

[2] Benjamin Wallenborn Dirk Frischalowski. Einführung in java-webanwendungen. *Uni Hagen*. URL http://www.isdb.fernuni-hagen.de/wbt/files/demo/jsp/JSP/Kursseite_35130.htm.

[3] Sebastian Springer. Vue.js: Das framework im überblick. *PHP Magazin, entwickler.de*, 2017. URL https://entwickler.de/leseproben/framework-ueberblick-vuejs-579760880.html.

[4] Google Trends. Trendsvue, 2018. URL https://trends.google.de/trends/explore?date=today%205-y&q=vue.js,react.js,angular.js.

[5] Ed Zynda. Getting started with vue router. *scotch.io*, 2017. URL https://scotch.io/tutorials/getting-started-with-vue-router.

[6] Joshua Bemenderfer. Introduction to routing in vue.js with vue-router. *Alligator.io*, 2017. URL https://alligator.io/vuejs/intro-to-routing/.

[7] Unknown, . URL https://www.thefreedictionary.com/Templating.

[8] Unknown. Vue.js templating. *Alligator.io*, 2016-2017. URL https://alligator.io/vuejs/templating/.

[9] Anthony Gore. Exploring vue.js: Reactive two-way data binding. *medium.com*, 2016. URL https://medium.com/js-dojo/exploring-vue-js-reactive-two-way-data-binding-da533d0c4554.

[10] Opher Etzion and Peter Niblett. *Event processing in action*. Manning, Greenwich, Conn., 2011. ISBN 1935182218. URL http://proquest.tech.safaribooksonline.de/9781935182214.

[11] Unknown. Event handling, 2018. URL https://vuejs.org/v2/guide/events.html.

[12] Unknown. Form validation, . URL https://vuejs.org/v2/cookbook/form-validation.html.

[13] Unknown. Components, . URL https://v1.vuejs.org/guide/components.html#What-are-Components.

[14] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. A survey on reactive programming. *ACM Computing Surveys*, 45(4): 1–34, 2013. ISSN 03600300. doi: 10.1145/2501654.2501666.

[15] Christian Lambert and Matthias Ebbing. Reaktiv in java mit rxjava: Reaktiv in die praxis: Reaktive programmierung mit rxjava, 2016. URL https://jaxenter.de/reaktiv-in-die-praxis-reaktive-programmierung-mit-rxjava-39088.

[16] Jan Carsten Lohmüller. Reactive programming – mehr als nur streams und lambdas, 2016. URL https://www.informatik-aktuell.de/entwicklung/programmiersprachen/reactive-programming-mehr-als-nur-streams-und-lambdas.html.

[17] Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson. Das reaktive manifest, 2014. URL https://www.reactivemanifesto.org/de.

[18] Paul Kögel. React einsteiger tutorial, 2015. URL http://reactjs.de/posts/react-tutorial.

[19] Marcin Skirzynski. Virtuelles dom mit react.js, 2015. URL http://reactjs.de/posts/virtuelles-dom-mit-react-js.

[20] Unknown. Comparison with other frameworks, 2018. URL https://vuejs.org/v2/guide/comparison.html.

[21] Toni Haupt. Vue.js – it's simple until you make it complicated, 2018. URL https://blog.codecentric.de/2018/02/vue-js-simple-make-complicated/.

[22] Kevin Peters. Large-scale vuex application structures, 2017. URL https://medium.com/3yourmind/large-scale-vuex-application-structures-651e44863e2f.

[23] Svensson Alexander. Speed performance comparison of javascript mvc frameworks, 2015. URL http://www.diva-portal.org/smash/get/diva2:998701/FULLTEXT03.pdf.

[24] Naveen Pete. Mvc, mvvm and angular. 2016. URL https://naveenpete.wordpress.com/2016/09/07/mvc-mvvm-and-angular/.

[25] Unknown. The mvvm pattern. *Microsoft*, 2012. URL https://msdn.microsoft.com/en-in/library/hh848246.aspx?irgwc=1&OCID=AID681541_aff_7593_1211691&tduid=(ir_VqY0n%3AR2F0Iazt90ZWSkDWZQUkjwTGy5NQ1xQg0)(7593)(1211691)()(ir)&clickid=VqY0n%3AR2F0Iazt90ZWSkDWZQUkjwTGy5NQ1xQg0&ircid=7593.

[26] Norbert Eder. Mvvm das model. 2017. URL https://www.norberteder.com/mvvm-das-model/.

[27] Norbert Eder. Mvvm die view. 2017. URL https://www.norberteder.com/MVVM-Die-View/.

[28] Norbert Eder. Mvvm das viewmodel. 2017. URL https://www.norberteder.com/MVVM-Das-ViewModel/.

[29] Lukas Jaeckle, Joachim Goll, Manfred Dausmann. Das architekturmuster model-view-viewmodel. 2015.

[30] Saisang Cai. Datenbindung. *Microsoft*, 2017. URL https://docs.microsoft.com/de-de/dotnet/framework/wpf/data/data-binding-wpf.

[31] Unknown. Mvvm tutorial. *tutorialspoint*, . URL https://www.tutorialspoint.com/mvvm/mvvm_advantages.htm.

[32] Artem Syromiatnikov and Danny Weyns. A journey through the land of model-view-design patterns. In *2014 IEEE/IFIP Conference on Software Architecture*. IEEE, 2014. doi: 10.1109/wicsa.2014.13.

[33] A. Leff & J.T. Rayfield, editor. *Web-application development using the Model/View/Controller design pattern*, 2001. IEEE Comput. Soc.

[34] Rui Peres. Model-view-controller (mvc) in ios: A modern approach. 2016. URL https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach.

[35] Alexander Schatten, Stefan Biffl, Markus Demolsky, Erik Gostischa-Franta, Thomas Östreicher, Dietmar Winkler. Delegation pattern. *Best-Practise Software Engineering*, 2013. URL http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/delegation.html.

[36] Leif Singer. Model-view-controller. *Uni Hannover*, 2004. URL http://www.se.uni-hannover.de/priv/lehre_2004winter_seminar_software_entwurf/04_mvc.pdf.

[37] Edward Curry and Paul Grace. Flexible self-management using the model-view-controller pattern. *IEEE Software*, 25(3):84–90, 2008. doi: 10.1109/ms.2008.60.

[38] John Deacon. Model-view-controller (mvc) architecture. *JOHN DEACON Computer Systems Development, Consulting & Training*, 1995, 2000, 2005.

[39] Matti Bragge. Model-view-controller architectural pattern and its evolution in graphical user interface frameworks, 2013. URL http://www.doria.fi/bitstream/handle/10024/92156/Model-View-Controller%20architectural%20pattern%20and%20its%20evolution%20in%20graphical%20user%20interface%20frameworks.pdf?sequence=2&isAllowed=y.

[40] Mike Potel. Mvp: Model-view-presenter. the taligent programming model for c++ and java. *Taligent, Inc*, 1996.

[41] R.Heini. Ziele und anforderungen: Funktionale, nicht funktionale anforderungen, 2010. URL http://www.anforderungsmanagement.ch/in_depth_vertiefung/ziele_und_anforderungen/index.html.

[42] Sophist. Anforderungen, 1998-2011. URL http://www.sophist.de/anforderungen/requirements-engineering/faq-requirements-engineering/.

[43] Joachim Goll. *Methoden und Architekturen der Softwaretechnik*. Studium. Vieweg+Teubner Verlag, Wiesbaden, 1. aufl. edition, 2011. ISBN 978-3-8348-1578-1. doi: 10.1007/978-3-8348-8164-9. URL http://dx.doi.org/10.1007/978-3-8348-8164-9.