

---

# 1 Vue im Überblick

Vue.js, englisch ausgesprochen [vjuːʃ], wie *View*, ist ein Framework zur Implementierung von Webanwendungen. Nicht selten wird Vue.js als Bibliothek bezeichnet, weil es sich durch den progressiven Ansatz, der in diesem Kapitel erläutert wird, gut in eigene Projekte jeglicher Größe und Komplexität eingliedert. Der Hauptunterschied zwischen einem Framework und einer Bibliothek ist, dass sich unser Code den Standards und Vorgaben eines Frameworks anpassen muss, während sich eine Bibliothek unseren Standards und Vorgaben anpasst. Da sich Vue.js bereits durch minimale Veränderungen im Projekt integrieren lässt, schwankt die Definition zwischen Framework und Bibliothek. Dieses Buch folgt der Kategorisierung als Framework, da es bei einem konsequenten Einsatz die Architektur der Webanwendung vorgibt – ein unverwechselbares Merkmal eines Frameworks. Im weiteren Verlauf wird Vue.js, je nach Version auch Vue.js 2 oder Vue.js 3 genannt, als Vue bezeichnet. Das *js* als Abkürzung für JavaScript ist im Sprachgebrauch nicht notwendig.

## 1.1 Was ist ein JavaScript-Frontend-Framework?

Bis in die frühen 2000er-Jahre hinein waren komplexe Webanwendungen im Browser undenkbar. Sowohl von der Performance her als auch vom notwendigen Tooling war an eine Entwicklung von Webanwendungen nicht zu denken. JavaScript war hauptsächlich im Einsatz, um kleinere Funktionalitäten auf Webseiten zu realisieren; weit entfernt vom heutigen Anwendungsfeld und den Möglichkeiten.

Geändert hat sich das mit der Einführung von Google Maps und Gmail im Jahr 2005 in den USA und ab 2006 auch als Webdienst in Deutschland. Zwei Anwendungen, die schon damals bei ihrer Einführung problemlos im Browser liefen – angetrieben durch asynchrone Ajax-Requests, die Anfragen über das Netzwerk möglich machten. Diese Möglichkeiten wurden in den folgenden Jahren konsequent ausgebaut. Browser haben neue Funktionen bekommen, ebenso wie neue und angepasste APIs. Zudem wurde JavaScript als Sprache und Webstandard grundlegend überarbeitet und zum Teil fast neu geschaffen. AngularJS, Ember, Knockout und Backbone.js gehörten zur ersten Welle an neuartigen und moder-

nen Web-Frameworks, um diese rasante Entwicklung von Webanwendungen zu unterstützen. Diese Geburtsstunde von Web-Frameworks hat dazu geführt, dass immer mehr Interaktionen mit dem Browser und dem Document Object Model (DOM) einer Webseite abstrahiert wurden. Das hatte neue Funktionen, bessere Performance und eine verkürzte Entwicklungszeit zur Folge.

Ein Frontend-Framework ist ein Gerüst für das Erstellen eines Frontend. Mit an Bord sind normalerweise Möglichkeiten, Dateien zu strukturieren, zum Beispiel um Komponenten und Daten mit DOM-Elementen zu verknüpfen. Ein Frontend-Framework hilft zudem bei der Trennung der Belange (*Separation of Concerns*), indem es sich ausschließlich um die Anzeige kümmert und Anfragen bezüglich Daten an ein Backend weiterleitet. Solche Frameworks bieten viele Vorteile, wie bessere Wartbarkeit, saubere Trennung der Belange, höhere Geschwindigkeit, eine gute Zusammenarbeit zwischen Entwicklern und eine breite Community. Es gibt aber auch Nachteile, wie die Einführung einer weiteren Abstraktionsschicht und die damit einhergehende erhöhte Komplexität, eine Lernkurve, die (zu) schnelle Evolution des Ökosystems, die Notwendigkeit, ein Projekt spezifisch nach den Wünschen des Frameworks aufzusetzen, und einen Overkill für kleinere Projekte.

Ein JavaScript-Web-Framework erlaubt es somit, moderne Anwendungen mit modernen Tools zu erschaffen, mittlerweile nicht mehr nur fürs Web im Browser, sondern ebenfalls auf dem Desktop und als mobile Anwendungen auf Smartphones. Wer sich bisher noch nicht sicher war, was ein JavaScript-Frontend-Framework oder Web-Framework bedeutet, hat jetzt hoffentlich einen ersten Einblick bekommen. Zudem ist Vue als Einstieg in die große Welt der Webentwicklung genau richtig.

## 1.2 Historie und das Team rund um Vue

Vue wurde von *Evan You* erschaffen, der bei Google mit AngularJS an einer Reihe von Projekten gearbeitet hat. In einem Interview sagte er mal, dass er das Beste an AngularJS herauslösen und als leichtgewichtiges Projekt anbieten wollte. Genau das hat er getan. Im Juli 2013 fanden die ersten Commits im Projekt statt. Eine erste öffentliche Version 0.6 erschien im Dezember des gleichen Jahres.

Seitdem ist die Beliebtheit von Vue konstant gewachsen, bis zur Version 3.0, die im September 2020 erschien. Im Jahr 2016 waren es circa 7.500 Sterne, im Jahr 2017 bereits über 36.000. Zudem zählte das Vue-Repository als das populärste Repository im Jahr 2016 (Stars, 2016). Das GitHub-Repository für Vue 2 (Team V., GitHub Repository für Vue 2, 2022) mit dem Quelltext des Vue-Projekts zählt mittlerweile über 190.000 Sterne und über 31.000 Forks (beides Stand Februar 2022). Da Vue 3 neuorganisiert wurde, gibt es dazu ein eigenes Repository (Vue, 2022). Das ist eine beachtliche Steigerung. Denn auch wenn Sterne auf GitHub keine verlässliche Metrik für die Beliebtheit sind, sind sie zumindest ein

Indikator. Eine aktuelle Auswertung in der Umfrage *The State of JS 2021* kommt zu dem Ergebnis, dass die Beliebtheit und Zufriedenheit etwas abgenommen hat (80%), aber immer noch auf einem hohen Wert verbleibt. Das Interesse und die Nutzung verbleiben bei einem nahezu unveränderten Wert (circa 50%) (State of JS Team, 2022).

Mittlerweile ist nicht mehr nur Evan You mit der Entwicklung von Vue beschäftigt. Eine ganze Reihe von Personen auf der ganzen Welt arbeitet gemeinsam am Projekt und den zahlreichen offiziellen Erweiterungen.

### 1.3 Was ist Vue und was zeichnet es aus?

Das Ziel von Vue (Team V., *Vue.js – The Progressive JavaScript Framework*, 2022) ist es, die Vorteile reaktiver Datenbindung (*Reactive Data Binding*) und kombinierbarer Komponenten (*Composable Components*) mit einer einfach zu nutzenden Schnittstelle zu verknüpfen und zur Verfügung zu stellen.

Diese einfache API und die damit verbundene niedrige Hürde für den Einstieg sind ein Merkmal, das Vue auszeichnet und von vielen anderen Web-Frameworks und -Bibliotheken abhebt. Denn andere Frameworks und Bibliotheken gehen oft den Weg, eine Schnittstelle anzubieten, die so umfassend wie möglich ist. Ein weiteres Merkmal ist, dass Vue nicht als das alles umfassende Web-Framework gedacht ist. Vue nutzt das *Model-View-ViewModel-(MVVM-)*Entwurfsmuster und fokussiert sich auf den View-Layer und nur auf diese Schicht – ein weiterer Punkt, warum Vue als einfach zu adaptieren gilt, wenn es um den Einsatz im eigenen Projekt geht. Diese Designentscheidungen führen zudem dazu, dass sich Integration und Kombination mit anderen Bibliotheken einfach umsetzen lassen. Vue sperrt sich dabei selten, und die Community hat eine umfassende Palette an guten Erweiterungen geschaffen, die gerne als First-Party-Erweiterungen bezeichnet werden.

Diese Merkmale bedeuten aber nicht, dass wir mit Vue keine umfangreichen oder anspruchsvollen Webprojekte stemmen können. Mit Vue lassen sich sehr wohl hochkomplexe *Single Page Applications (SPAs)* erzeugen, was die zahlreichen kurz vorgestellten Use Cases gegen Ende dieses Buchs beweisen.

### 1.4 Warum Vue?

Da mittlerweile zahlreiche Web-Frameworks existieren, steht immer die Frage im Raum, warum dieses oder jenes Framework für das eigene Projekt zum Einsatz kommen soll. In diesem Fall also die simple Frage: Warum Vue?

Das läuft auf die Frage hinaus, was Vue für einen konkreten Mehrwert bietet. Hier kommt der progressive Ansatz von Vue zum Tragen. Die Macher von Vue bezeichnen es als progressives Framework, weil Vue es erlaubt, mit minimalem Aufwand eine Anwendung mit Vue zu starten oder eine bereits bestehende mit ei-

ner ersten Vue-Komponente auszustatten. Dazu ist es nicht notwendig, die gesamte Architektur des Projekts von Grund auf neu zu entwickeln, damit es den Anforderungen von Vue gerecht wird. Die minimal notwendigen Schritte sind, Vue einzubinden, im einfachsten Fall als loses Skript, um dann eine Seite mit einer View-Komponenten auszustatten. Diese Änderungen als Grundlage genutzt, lassen sich weitere Komponenten einsetzen, Komponenten miteinander kombinieren und dergleichen. Wenn mit der Zeit die Anforderungen wachsen, lassen sich weitere Komponenten einbeziehen, zum Beispiel für das Routing oder das State Management.

Diese Möglichkeit, Vue in einem Projekt nach und nach einzusetzen, ist unter Entwicklerinnen und Entwicklern ein häufiger Grund, warum die Wahl auf Vue fällt. Insbesondere beim Einsatz des ersten Web-Frameworks ist diese iterative Adaption von Features ein großer Vorteil. Vue passt sich den Fähigkeiten der Entwicklerin oder des Entwicklers an und nicht andersherum. Das wirkt aktiv gegen den Trend vieler Frameworks und Bibliotheken, Neuankömmlinge abzuschrecken, weil sich das gesamte Projekt nach den Vorgaben des Web-Frameworks richten muss, was dazu führen kann, dass sich Einsteiger am Anfang komplett verloren fühlen. Vue wird daher manchmal als das neue jQuery bezeichnet, weil es ebenso einfach einsetzbar ist wie die einstige Vorzeigebibliothek für dynamische Webseiten.

Vue besitzt einige weitere Vorteile. Es ist klein, was die Downloadgröße anbelangt, besitzt das Prinzip der Single-File Components und damit eine gute Lesbarkeit, ein solides Tooling- und Ökosystem und gilt im Allgemeinen als einfach zu nutzen. Aber natürlich ist nicht alles eitel Sonnenschein. Mit Vue 3 wurde das Reactivity-System verbessert, es ist aber immer noch recht komplex. Außerdem besteht die Gefahr der Über-Flexibilisierung. Wenn Vue ohne große Hürden überall zum Einsatz kommen kann, kann es zu nicht gut wartbarem Code kommen. Hier ist etwas Disziplin gefordert.

## 1.5 Unterschiede zu anderen Web-Frameworks

Über die Jahre hat sich eine ganze Reihe von Web-Frameworks etabliert. Allen voran sind sicherlich Angular und React zu nennen, die eine umfassende Community und eine Vielzahl prominenter Webprojekte hinter sich vereinen. Vue muss sich allerdings nicht mehr hinter diesen Projekten verstecken. Insbesondere mit Vue 3 sind erhebliche Änderungen in das Projekt eingeflossen, sodass sich die drei großen Web-Frameworks Angular, React und Vue weiter annähern.

Beim Vergleich zwischen Angular und Vue stechen einige Punkte heraus. Das mögen von Projekt zu Projekt Gründe sein, um Vue einem Einsatz von Angular vorzuziehen. In der Regel ist der Einsatz eines Web-Frameworks von Fall zu Fall zu entscheiden. Vue ist beispielsweise deutlich einfacher zu nutzen als Angular, was primär am API-Design liegt. Der Einstieg in Vue gelingt häufiger einfacher

und schneller. Zudem ist Vue in vielen Fällen flexibler und bringt eine weniger starke eigene Meinung mit, wie ein Projekt zu strukturieren ist. Vue ist primär eine Schnittstellschicht (*Interface Layer*), sodass sich auf Webseiten einige Features von Vue nutzen lassen, ohne direkt alles umbauen und umstrukturieren zu müssen. Ein großer Unterschied zu Angular ist noch, dass Vue im Kern praktisch keine zusätzlichen Features mitbringt, zum Beispiel kein Routing. Der Ansatz von Vue ist die Annahme, dass in einem Web-Framework sowieso ein Bundler wie Webpack für externe Module zum Einsatz kommt. Ein weiterer Unterschied zwischen Angular und Vue ist, dass Vue im Standard auf ein One-Way-Data-Binding zwischen Eltern- und Kindkomponente setzt, wohingegen Angular Two-Way anbietet. Vue trennt zudem ganz klar zwischen Direktive sowie Komponente und hat vielfach die bessere Performance. Ein Grund dafür ist das transparente Tracking von Abhängigkeiten auf Basis eines asynchronen Queuing-Verfahrens. Alle Änderungen werden unabhängig voneinander getriggert, außer sie haben eine explizite Abhängigkeitsbeziehung. Viele dieser Probleme und Unterschiede wurden in vergangenen und werden in zukünftigen Versionen von Angular angepackt, so dass diese Unterschiede von Version zu Version kleiner werden.

React und Vue teilen sich viele Gemeinsamkeiten, da beide Web-Frameworks reaktive und kombinierbare Komponenten für den View-Layer anbieten. React nutzt für Änderungen das Virtual DOM, ein vormals großer Unterschied zu Vue. Mittlerweile implementiert Vue ebenfalls ein virtuelles DOM, eine Light-Version des realen DOM. Hier haben sich React und Vue deutlich angenähert. Aus Sicht des API wird es häufig kritisch gesehen, dass die Render-Funktionen in React viel Logik enthalten und somit nach und nach eher einem Programm gleichen, was sie letztendlich auch sind, anstatt sich nur um die visuelle Repräsentation zu kümmern. Das mag für Entwickler noch verträglich sein, für Designer kann es die Arbeit mit diesen Templates erschweren. Das React-Team verfolgt zudem das ambitionierte Ziel, React zu einem plattformunabhängigen UI-Entwicklungsparadigma zu entwickeln. Vue konzentriert sich im Gegensatz dazu darauf, eine einfache pragmatische Lösung für das Web zu bieten. React arbeitet zudem gut mit funktionalen Mustern (Patterns) zusammen, was gerade zu Beginn zu einer etwas steileren Lernkurve und mehr Hürden führen kann. Sind diese funktionalen Muster aber bekannt, kann React seine ganze Stärke ausspielen. Vue gilt als gemeinhin einfacher für den Einstieg. Mit Vue 3 und dem Composition API, ebenfalls Thema in diesem Buch, geht aber auch das neue Vue einen deutlich funktionaleren – aber optionalen – Weg. Ein großer Vorteil von React ist das ausgezeichnete State Management. Mit Flux/Redux ((Facebook, 2022) und (Abramov, Redux, 2022)) werden Möglichkeiten angeboten, die lange Zeit allen anderen Frameworks voraus waren. Vue setzt hier, getreu dem Designprinzip des Web-Frameworks, auf externe Bibliotheken wie Vuex, und auch Redux lässt sich mit Vue kombinieren. Als letztes Merkmal ist der Trend bei React zu nennen, alles in JavaScript einzubetten, auch die CSS-Anweisungen. Vue löst das mit dem Feature der Single-File

Components, in denen das CSS pro Komponente gekapselt ist, aber als reines CSS bereitsteht. Dadurch können zum Beispiel weiterhin Präprozessoren für CSS zum Einsatz kommen.

Vue grenzt sich auch von Frameworks wie Ember, Polymer und Riot in vielen Aspekten ab. In Ember ist die Reaktivität beispielsweise nur über eigene Ember-Objekte herzustellen, zudem ist die Template-Syntax von Vue einfacher zu nutzen, da sich JavaScript-Anweisungen integrieren lassen. Zudem ist Vue noch immer ein ganzes Stückchen performanter. Bei Polymer ist zum Beispiel ein Nachteil, dass das Framework auf dem aktuellen Feature der Web Components fußt. Für Browser, die das nicht unterstützen, sind umfangreiche und nicht-triviale Polyfills notwendig. Im Vergleich zu Riot gibt es eine ganze Reihe von Unterschieden. Riot rendert zum Beispiel alle if-Verzweigungen immer und blendet je nach Bedingung dann lediglich die nicht benötigten aus. Dadurch entstehen Performance-Bottlenecks. Zudem sind das Tooling und die externen Module, wie das Routing, in Vue deutlich fortschrittlicher und mächtiger.

Dieser Überblick kann nur ein Ausschnitt aus der Welt der Web-Frameworks darstellen, da sich das Web und die Technologien darin beständig weiterentwickeln. Für einen vollständigen Überblick gibt es zu viele Frameworks und Bibliotheken, und die Geschwindigkeit, mit der Änderungen eingebracht werden, ist schwindelerregend.

## 1.6 Vue 2 oder Vue 3 lernen?

Wer gerade anfängt, Vue kennenzulernen, sollte direkt mit der Version 3 starten. Das ist diejenige, die in Zukunft weiterentwickelt wird und die neuen Konzepte inklusive vieler Verbesserungen nutzt. Zudem sind zahlreiche Funktionalitäten optional, lassen sich also gut graduell einsetzen.

In der Praxis ist die Entscheidung aber gar nicht so leicht. Es hängt zum Beispiel davon ab, ob in einem Team gearbeitet wird, das Vue 2 verwendet und keine Pläne für eine Migration auf Vue 3 hat. Es hängt auch vom eigenen Zeitplan ab und ob die eigene App Abhängigkeiten benötigt, die noch nicht mit Vue 3 kompatibel sind. Das wird sich zwar über die Zeit aller Voraussicht nach lösen, aber in solchen Situationen ergibt es vielleicht keinen Sinn, die neueste Syntax und APIs von Vue 3 zu lernen, wenn diese nicht in der täglichen Arbeit zum Einsatz kommen können. Bis auf Mixins kann Vue 3 so eingesetzt werden, wie es von Vue 2 bekannt ist. Es gibt daher keinen triftigen Grund, noch mit Vue 2 zu starten, wenn die in Vue 3 fehlenden Mixins kein Argument sind.

Der Einstieg mit Vue 3 ist also keine schlechte Idee und zukunftssicher. So ist sichergestellt, dass wir nicht in Kürze abgehängt sind, weil Vue 2 dann doch schneller ausläuft als angenommen.

## 1.7 Der Blick in die Zukunft

Auch wenn die Veröffentlichung von Vue 3 und der Wechsel zur Version 3 als offizielle Version im Februar 2022 zwei Events zum Feiern im *Vueniverse* waren, so blieb die Entwicklung danach natürlich nicht einfach stehen. In Zukunft sind weitere neue Features geplant.

Dazu gehört *Reactivity Transform*, der finale Baustein für das Reactivity-System und das Script Setup. Damit wird es deutlich einfacher, reaktive Daten zu erstellen, und es verkleinert die Einstiegshürde für alle, die von Vue 2 kommen.

Auch im Ökosystem sind noch einige Änderungen zu erwarten. Zum Beispiel die stabile Version von Nuxt 3 mit zahlreichen Änderungen im Gepäck. So ähnlich sieht es bei Vuetify aus, einem populären Framework basierend auf dem Material Design. Das Team rund um Vuetify arbeitet aktuell an der finalen Unterstützung von Vue 3.

## 5 Das (neue) Reactivity-System

Mit *Reactivity* wird ein Kernkonzept zusammengefasst, das den Unterschied von modernen Web-Frameworks zu reinem JavaScript oder Bibliotheken wie jQuery verdeutlicht. Es ist der Unterschied zwischen deklarativem und imperativem Rendering. In Vue sind die Template-Syntax und das virtuelle DOM zwei Kernaspekte davon. In jQuery, um bei dem Beispiel zu bleiben, müssen Änderungen an der Oberfläche ganz explizit gesetzt werden. Eine Funktion ändert direkt einen Wert im DOM, fügt eine CSS-Klasse hinzu oder löscht ein Kindelement. Das nachfolgende Code-Snippet zeigt das an einem Beispiel, bei dem eine CSS-Klasse zu DOM-Elementen hinzugefügt wird, wenn ein Button-Klick erfolgt.

```
$('#button').click(function() {  
    $('#h1, p').addClass('important');  
});
```

Das sind alles direkte, imperative Anweisungen in Form von Kommandos, die das *Wie* darstellen. Bei der deklarativen Sichtweise moderner Web-Frameworks werden Daten verändert, die einen Zustand wiedergeben. Durch diese Zustandsänderungen verändern sich beispielsweise Resultate von Berechnungen, die Inhalte oder die Anzahl von Listenelementen oder sonstige Eigenschaften von View-Elementen. Auf Basis dieser Datenänderungen wird anschließend der View Layer angepasst. Wir Entwicklerinnen und Entwickler haben damit deklarativ bestimmt, *was* geschehen soll. Das *Wie* interessiert uns an dieser Stelle zunächst nicht.

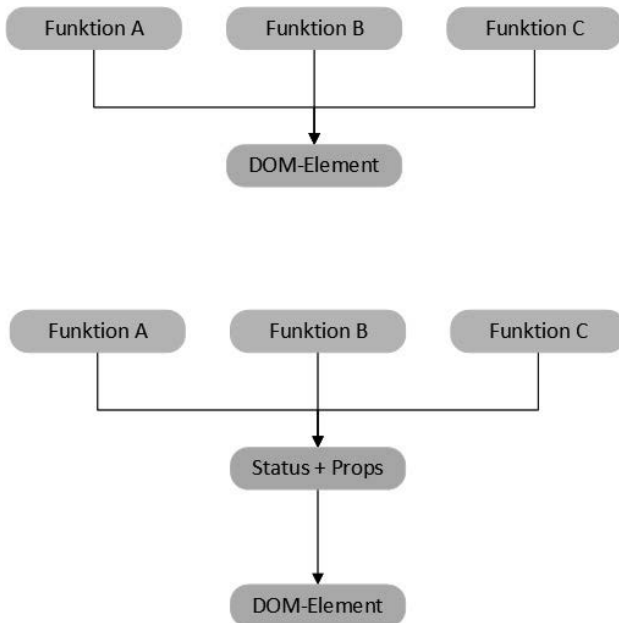
Zusammengefasst bedeutet Reactivity, dass sich das Resultat einer Berechnung, wie auch immer diese ausgestaltet ist, ändert, wenn sich die zugrunde liegenden Daten ändern. Die veränderten Daten lösen somit einen gewollten Seiteneffekt aus. Das ist immer dann sinnvoll, wenn Daten, Informationen oder eben Resultate von Berechnungen in der Benutzeroberfläche angezeigt werden: zum Beispiel bei einer HTML-Seite in einer Webanwendung. Anders formuliert bietet das Reactivity-System einen Mechanismus, die Daten im Modell automatisch synchron mit der Datenrepräsentation, dem View-Layer, zu halten. Änderungen an den Daten werden automatisch in der View wiedergegeben, weil entsprechender Code von Vue ausgeführt wird. Das *Was* (semantisch) ist der bestimmende Faktor, nicht das *Wie* (technisch). Die Abbildung 5–1 visualisiert diesen Unterschied schematisch.



Die Reaktivität von Daten ist ein fundamentales Feature moderner Web-Frameworks. JavaScript als Programmiersprache ist im Standard nicht reaktiv, wie das nachfolgende Beispiel verdeutlicht:

```
let x = 3;  
let y = 5;  
let result = x + y;  
  
console.log(result); // 8  
  
y = 3;  
  
console.log(result); // Weiterhin 8
```

Die Ausgabe auf der Konsole ist hier vergleichbar mit einem View-Layer, also zum Beispiel der Anzeige in einer HTML-Seite. In modernen Webanwendungen muss sich das Resultat der Berechnung verändern, wenn sich die zugrunde liegenden Daten ändern, was aber hier nicht passiert. Eine einmal im DOM gerenderte Information, zum Beispiel durch eine Variable, ändert sich nicht auf magische Weise, wenn sich der Inhalt der Variablen ändert. Genau das ist aber eine primäre Anforderung moderner, datengetriebener Webanwendungen. Dieses *Zustandsmanagement (State Management)* manuell zu implementieren und die Daten mit der View synchron zu halten, ist zwar möglich, bedeutet aber einen erheblichen Aufwand, was Zeit, Nerven und Geld kostet sowie zusätzlich fehleranfällig ist.



**Abb. 5-1** Schematische Darstellung eines imperativen (oben) und deklarativen (unten) Änderungsprozesses von Funktionen an einem DOM-Element. Im deklarativen Fall wird das Element indirekt über Änderungen an den Daten verändert.

## 5.1 Der Ansatz von Vue

Das Reactivity-System von Vue ist eines der zentralen Merkmale des Web-Frameworks. Ein Reactivity-System muss primär drei Aufgaben erfüllen, damit es einen Nutzen entfaltet:

1. Es muss erfassen (tracken), wenn ein Wert gelesen wird. Damit wird ermittelt, welcher Code welche Abhängigkeiten auf Variablen und damit auf Daten hat. Beim Ändern von Daten müssen diese Codeteile erneut ausgeführt werden, damit Änderungen an den Daten über die reine Änderung der Variablen hinweg wirksam werden.
2. Das System muss erfassen, wann Werte verändert werden. Denn das ist der Ausgangspunkt für reaktive Veränderungen in anderen Bereichen der Anwendung.
3. Es ist notwendig, dass der Code neu ausgeführt wird, der Daten einer Variablen gelesen hat, die anschließend verändert wurde. Das ist der Punkt, in dem sich andere Codeteile einer Anwendung, zum Beispiel Elemente im View-Layer, anpassen und das Reactivity-System auch für uns Entwicklerinnen und Entwickler sowie für die Nutzerinnen und Nutzer sichtbar wird.

Das Reactivity-Modell von Vue wird gerne als unaufdringlich oder unauffällig bezeichnet. Das bedeutet, es fällt bei der täglichen Entwicklung nicht auf, dass es überhaupt da ist. Es funktioniert einfach. Das Modell einer Vue-Instanz, der Datenbereich, besteht aus einfachen JavaScript-Objekten. Die Eigenschaften dieser Objekte enthalten die Daten, und Änderungen daran werden automatisch erfasst, damit Vue das View anpassen kann. Das bedeutet aber auch, dass alle Daten, die reaktiv sein sollen, im Datenbereich einer Vue-Instanz oder einer Komponente definiert werden müssen. Fehlt dort zum Beispiel eine Variable, die dann später mit der Textinterpolation gerendert werden soll, warnt Vue, dass die Variable unbekannt ist. Ohne die vorherige Definition weiß Vue nichts von der Variablen und konnte diese nicht ins Reactivity-System einbinden und mit den notwendigen Anpassungen ausstatten. Wie die dafür nötigen Anpassungen aussehen, beschreibt der nachfolgende Abschnitt 5.2 im Detail auch mit zusätzlichen Informationen für alle, die das Reactivity-System aus Vue 2 kennen. Denn zwischen beiden Versionen bestehen erhebliche Unterschiede. Dieses automatische Reactivity-System hat einige Vorteile. Es spart Zeit bei der Entwicklung, der Code wird einfacher und es reduziert unseren kognitiven Aufwand.

Um das Ganze an einem Beispiel festzumachen: In Vue reicht es aus, einen Datenbereich zu erstellen, dort Eigenschaften zu definieren und diese im HTML-Template zu nutzen. Ändern sich die Daten, ändert sich auch die Anzeige im HTML-Template. Das folgende Beispiel zeigt einen Ausschnitt aus der Definition eines Datenbereichs und einer Methode sowie das zugehörige Template.

```

data() {
  return {
    message: 'Ich bin ein Test! ',
  };
},
methods: {
  onChangeMessage() {
    this.message = 'Ich bin die neue Nachricht! ';
  },
},
<div>
  <span>Reaktive Daten, die sich verändern: {{ message }}</span>
  <br />
  <button @click="onChangeMessage">Text verändern</button>
</div>

```

In diesem Fall registriert Vue, dass es eine Eigenschaft `message` gibt, die in den Daten definiert ist. Die Methode `onChangeMessage` greift darauf zu, um sie bei einem Methodenaufruf einmalig auf einen neuen Wert zu verändern. Gebunden werden diese Informationen über das Reactivity-System im Template in einem `span`-Element, um den Text möglichst einfach anzuzeigen. Abbildung 5–2 zeigt die Veränderung bei einem einfachen Button-Klick. An diesem einfachen Beispiel wird deutlich, wie mächtig das Reactivity-System ist, bei gleichzeitig niedrigschwelligem Einsatz. Daten definieren, nutzen und verändern, reicht völlig aus. Dabei ist noch der Hinweis relevant, dass der Mausklick, der die Methode aufruft, nur ein Beispiel von vielen ist. Durch welchen Mechanismus die Daten verändert werden, kümmert Vue nicht. Der relevante Aspekt ist, dass sie verändert werden und dass alle davon abhängigen Teile der Anwendung auf diese Anpassung reagieren.

Reaktive Daten, die sich verändern: Ich bin ein Test!

Reaktive Daten, die sich verändern: Ich bin die neue Nachricht!

**Abb. 5–2** Reaktive Änderungen durch einen Button-Klick

Vue bietet drei Möglichkeiten, um Daten in HTML-Templates zu rendern: Property, Computed Property und Methode. Näheres dazu verrät das Kapitel 6 zu den Komponenten sowie das Kapitel 7, wenn es um die Verarbeitung von Eingabedaten geht.

Die Updates des DOM, die aus dem Reactivity-System erfolgen, werden von Vue *asynchron* durchgeführt. Intern wird eine Queue verwaltet, um alle Ände-

rungen von Daten in einem Event-Loop zwischenspeichern. Das verhindert unnötige Updates des DOM. Im nächsten Event-Loop-Tick werden die Änderungen dann am DOM durchgeführt. Da die Updates des DOM asynchron erfolgen und von Vue bis zum nächsten Tick zwischengespeichert werden, können wir über `nextTick` auf den nächsten Flush der DOM-Updates warten:

```
    this.$nextTick(() => {  
      // ...  
    });  
  
    await this.$nextTick();
```

Wir können entweder einen Callback als Argument angeben oder mit `await` auf den Promise warten.

## 5.2 Die Implementierung als Proxy in Vue 3

Damit das Reactivity-System in Vue funktioniert und um der Tatsache gerecht zu werden, dass es so wenig wie möglich im Weg steht, laufen im Hintergrund von Vue einige Prozesse ab, um aus normalen JavaScript-Objekten reaktive Vue-Objekte zu machen. Dieser Ablauf und die dazu notwendige Implementierung wurden erheblich zwischen den Versionen Vue 3 und 2 angepasst. Für Vue 3 wurde das Reactivity-System als Teil der Modularisierungsstrategie aus dem Vue Core entfernt und als eigenständiges Paket veröffentlicht. Der Code befindet sich wie gewohnt auf GitHub (Team V., @vue/reactivity GitHub Repository, 2022).

Vue 3 setzt in der neuen Umsetzung auf *Proxy-Objekte*. Diese sind notwendig, denn JavaScript besitzt im Standard keinen Mechanismus, um mitzubekommen, ob, geschweige denn wann, eine lokale Variable durch einen neuen Wert überschrieben wurde.

Die Zuweisung einer Variablen mit neuen Daten muss Vue aber abfangen, um anschließend darauf reagieren zu können. Daher werden diese Informationen zu Objekten, bei Vue 3 die angesprochenen Proxys, umgewandelt, weil eine Änderung an Eigenschaften eines Objekts erfasst werden kann.

### So läuft es in Vue 2

In Vue 2 wurden dazu noch die Daten und Objekte im Data-Bereich einer Komponente durchgegangen, um Getter- und Setter-Methoden für die Daten zu erstellen. Dadurch lassen sich die Zugriffe nachverfolgen.

Die Proxy-Implementierung nutzt die neuen Möglichkeiten von ECMAScript 6 (Mozilla, Proxy-Implementierung, 2022). Das dort eingeführte Proxy-Objekt erlaubt es, ein Proxy für ein anderes Objekt zu erzeugen, um dann die darauf ausgeführten Operationen abzufangen. Genau das, was Vue braucht, um einige der ärgerlichen Probleme mit dem Reactivity-System aus Vue 2 auszumerzen – mehr dazu in Abschnitt 5.3. Das Proxy-Objekt ist dafür zuständig, die lesenden und schreibenden Zugriffe auf das originale Objekt abzufangen und zu tracken. Das folgende Snippet zeigt vereinfacht, wie diese Proxys in Vue zum Einsatz kommen:

```
const person = {
  firstname: 'Fabian',
};

const handler = {
  get(target, property, receiver) {
    console.log('Lesender Zugriff... ');
    track(target, property);
    return Reflect.get(...arguments);
  },

  set(target, property, value, receiver) {
    console.log('Schreibender Zugriff... ');
    trigger(target, property);
    return Reflect.set(...arguments);
  },
};

const proxy = new Proxy(person, handler);
console.log(proxy.firstname);
```

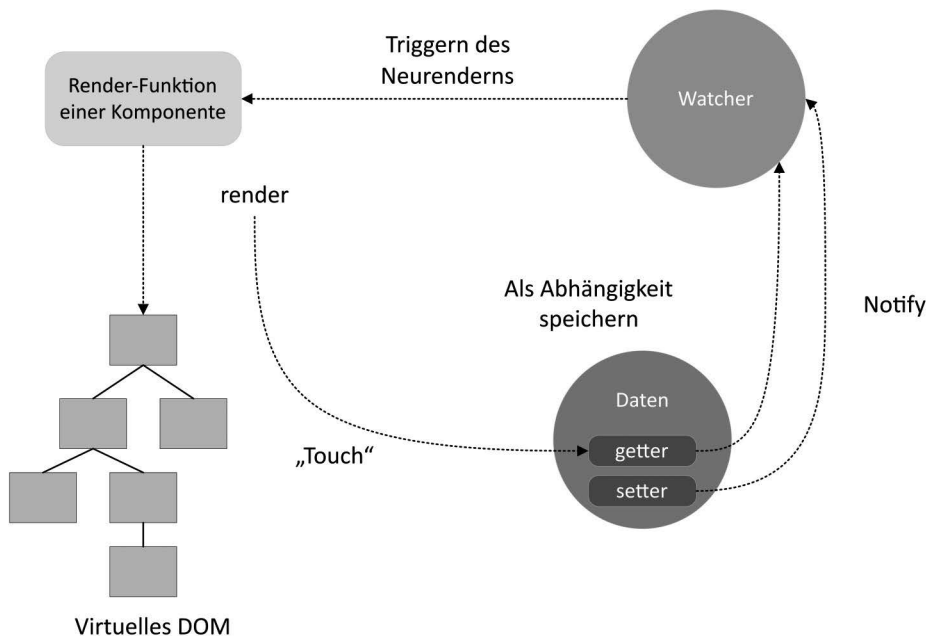
Das originale Objekt heißt in diesem Fall `person`. Zusätzlich wird ein Handler erstellt, der dann beim Erzeugen des Proxys zusammen mit dem originalen Objekt übergeben wird. Da der Handler eine `get`-Methode besitzt, wird diese nun aufgerufen, wenn die Eigenschaft gelesen wird. Es erfolgt somit die Ausgabe

```
Lesender Zugriff...
Fabian
```

auf der Konsole. Vue ist somit in der Lage, festzustellen, wenn die Daten gelesen werden. Durch die `set`-Methode wird auch das Schreiben neuer Daten über den Proxy erfasst. Vue nutzt das, um die lesenden Zugriffe über einen Aufruf der `track`-Methode zu tracken und die schreibenden Zugriffe über die `trigger`-Methode mitzubekommen. Über die `track`-Methode wird erfasst, dass die Eigenschaft eine Abhängigkeit zu einem bestimmten Codefragment ist. In Vue wird das ein Effekt genannt. Die `trigger`-Methode sorgt dafür, dass alle Codeteile erneut ausgeführt werden, wenn sich die Daten verändern. Diese Codeteile sind eben jene, die als abhängiger Code zu einer Variablen erkannt wurden.

Reflect (Mozilla, Reflect, 2022) sorgt hier dafür, dass sich das Binding von `this` korrekt verhält. Gewünscht ist, dass alle Methoden an den Proxy gebunden werden und nicht an das originale Zielobjekt. Dadurch ist sichergestellt, dass alle Zugriffe korrekt über den Proxy abgefangen werden können.

Abbildung 5–3 zeigt als schematische Darstellung, wie Änderungen nachverfolgt werden, um Änderungen zu triggern. Die Render-Funktion einer Komponente ist vereinfacht gesagt für den Aufbau des virtuellen DOM zuständig. Wenn die Komponente gerendert wird, werden alle Zugriffe auf Getter-Methoden erfasst, als sogenannter *Touch* bezeichnet. Diese Zugriffe werden im Watcher, pro Komponente wird ein Watcher erstellt, als Abhängigkeit erfasst. Denn diese per Getter angefragten Daten werden ja für das Rendern der Komponenten benötigt. Durch einen Zugriff auf eine Setter-Methode, wenn Daten verändert werden, lässt sich ein Notify an den Watcher absetzen. Da Daten geändert sind, ist ein neues Rendering erforderlich. Der Watcher weiß, welche Abhängigkeiten es gibt, und kann die betreffenden Codebereiche neu anstoßen und dann das Rendering durchführen. Dieser Vorgang passiert bei jedem Rendering beziehungsweise bei jeder Änderung an den Daten.



**Abb. 5–3** Schematische Darstellung, wie in Vue Änderungen nachverfolgt werden, um Änderungen zu erfassen und Komponenten neu zu rendern. Das Tracking von Änderungen ist ein Kern des Reactivity-Systems.

Damit erfüllt die Proxy-Implementierung die drei Kriterien an Reactivity-Systeme aus Abschnitt 5.1. Die Track-Funktion in einer Get-Methode erfasst die aktuelle Eigenschaft und den laufenden Effekt. Der Set-Handler bekommt mit, wenn ein Wert verändert wird. Die Trigger-Funktion findet heraus, welche Effekte auf der veränderten Eigenschaft basieren und kann diese erneut ausführen.

### Reactivity-System in Vue 2

In Version 2 von Vue ist das Reactivity-System komplett anders implementiert. Anstatt Proxys nutzt Vue 2 automatisch generierte Eigenschaften für Getter und Setter. Wenn ein JavaScript-Objekt in einer Vue-Instanz im Data-Bereich genutzt wird, geht Vue das Objekt durch und konvertiert alle Eigenschaften zu Getter und Setter (mit `Object.defineProperty`). Diese neuen Getter und Setter sind zwar für den Entwickler zunächst nicht sichtbar, werden aber unter der Haube für alle Änderungen an den Eigenschaften genutzt. Dadurch lassen sich beispielsweise Events auslösen, die über Änderungen informieren. Diese Implementierung ist der Grund, warum Vue 2 den IE8 und niedriger von Anfang an nicht unterstützt hat, denn dieses Feature kann nicht durch einen Patch (shim) nachgerüstet werden.

### 5.3 Probleme bei der Reaktivität

In Vue 3 kann die Performance beim Umwandeln normaler JavaScript-Objekte zu Proxy-Objekten zu einem Sorgenkind werden. Weil diese Umwandlungen rekursiv durchgeführt werden. Es betrifft somit auch alle verschachtelten Objekte. Das kann bei einer umfassenden Objektstruktur stark auf die Laufzeit schlagen. Insbesondere, wenn sich Objekte von Drittanbieter-Komponenten in dieser Objektstruktur befinden, da diese Objekte wiederum an sich bereits umfangreich sein können. Daher ist es wichtig, darauf zu achten, ob die Umwandlung mit Proxy-Objekten wirklich über die gesamte Objektstruktur hinweg notwendig ist. Über die Mechanismen von `markRaw` und den `shallow`-Funktionen besteht die Möglichkeit, selektiv aus dieser tiefen Umwandlung zu reaktiven oder Read-only-Objekten auszusteigen, also diese nicht durchzuführen. Im Zustandsgraphen einer Vue-Instanz lassen sich damit normale JavaScript-Objekte einbinden.

## Vue 2 und reaktive Objekte

In Vue 2 gab es zudem Probleme mit Objekten, wenn dort dynamisch Eigenschaften hinzugefügt oder entfernt wurden. Da die Kapselung mit Gettern und Settern nur bei der Definition des Objekts geschieht, werden nachträgliche Änderungen nicht erfasst. Bei einem Objekt, bei dem vorher keine Eigenschaft mit dem Namen `b` existierte, lässt sich daher über die folgende Zeile keine Eigenschaft hinzufügen, die Vue über das Reactivity-System überwacht.

```
const einObject.b = 12;
```

Bei Arrays gibt es ebenfalls Probleme, wenn Elemente direkt über den Index gesetzt werden oder die Länge des Arrays angepasst wird:


```
this.elements[10] = 99;  
this.elements.length = 1;
```

In beiden Fällen bekommt Vue 2 diese Änderungen nicht mit. In allen Fällen lässt sich die `Vue.set`-Methode nutzen, um Vue direkt mitzuteilen, dass Änderungen an den Objekten beziehungsweise Arrays vorliegen. Beispielsweise mit der folgenden Zeile Code:

```
Vue.set(vm.elements, indexOfItem, newValue);
```

Das funktioniert für das Beispiel mit dem Objekt und dem Array. Alternativ steht der Alias `vm.$set` zur Verfügung. Mit Vue 3 gehören diese Probleme aber erfreulicherweise alle der Vergangenheit an.

Ein weiteres Problem kann bei umfangreichen Objekten entstehen, die durch Vue 3 automatisch in Proxys umgewandelt werden. Dabei werden die Objekte *deep reactive*. Das bedeutet, dass die verschachtelten Daten ebenfalls reaktiv sind. Dies kann auch aus Versehen passieren, wenn wir ein Objekt in ein anderes einfügen und das eingefügte Objekt dadurch reaktiv machen. Über die Funktion `markRaw` markieren wir ein Objekt so, dass es niemals reaktiv wird.

Diese Leseprobe haben Sie beim  
 [edv-buchversand.de](https://edv-buchversand.de) heruntergeladen.  
Das Buch können Sie online in unserem  
Shop bestellen.  
[Hier zum Shop](#)