Einführung in Vue.js - Teil 2

Vue.js Tutorial: So entwickelt man Komponenten mit Vue.js

Marc Teufel 2 Jahren online 2 Kommentare

Im ersten Teil dieses Tutorials ging es darum, wo Vue.js herkommt und wie es einzuordnen ist. In diesem zweiten Teil wollen wir uns mit dem Komponentenmodell von Vue.js beschäftigen und lernen, wie man mit Single File Components ein wertvolles Hilfsmittel an die Hand bekommt, um wart- und erweiterbaren Quellcode zu schreiben.

Einführung in Vue.js – Teil 2

Egal ob man das Glück hat, ein ganz neues Projekt beginnen zu dürfen, oder die großartige, viel spannendere Herausforderung bewältigen darf, ein bestehendes und gewachsenes Projekt weiterzuführen – zu Beginn sind immer Entscheidungen zu treffen. Eine nicht ganz unwesentliche Entscheidung ist, wie man seinen Werkzeugkasten bestückt.

Die Programmiersprache dürfte in den meisten Fällen bereits gesetzt sein, trotzdem drängen sich Fragen auf: Wie schreibe ich meinen Quellcode, oder noch wichtiger, wie organisiere ich ihn? Wie überführe ich meine fachlichen Anforderungen in möglichst saubere, qualitativ hochwertige und gut wartbare Quellen, die von jedem Entwickler auch noch leicht zu überblicken und zu verstehen sind? In diesem Zusammenhang helfen uns Frameworks weiter. Vue.js [1] ist ein solches Framework, noch dazu ein vielseitiges und universelles.

Einführung in Vue.js: Das Tutorial zum Mitmachen

Teil 1: Einordnung und Überblick

Teil 2: Komponenten mit Vue.js entwickeln

Warum Vue.js einsetzen?

Die Vielseitigkeit von Vue.js äußert sich zunächst darin, dass man sich relativ schnell einarbeiten kann. Jeder Webentwickler, der Erfahrung mit HTML, CSS und JavaScript hat, wird sich rasch mit Vue.js vertraut machen können. Expertenwissen ist nicht erforderlich. Bekannte und weit verbreitete, aber doch mit einem Touch Komplexität vorbelastete JavaScript-Erweiterungen wie TypeScript oder JSX lassen sich zwar auch mit Vue.js verwenden, doch Vue.js setzt keine dieser Erweiterungen zwingend voraus. Die Idee hinter

Vue.js ist es, einfach zu bleiben. Getreu dem Motto „Schuster, bleib bei deinen Leisten!“ verschreibt sich Vue.js hier ganz der Programmiersprache JavaScript.

Das hat den angenehmen Nebeneffekt, dass man mit Vue.js seine Anwendungen noch unkomplizierter testen kann. Denn hier ist fast alles pures JavaScript, und mit Testframeworks wie Mocha [2] oder Jest [3] lässt sich ohne Klimmzüge nahezu alles einfach testen. Mehr noch: Da Vue.js auf den Kerntechnologien HTML, CSS und JavaScript aufbaut, lässt es sich unkompliziert in bestehende Projekte einbauen. So lassen sich einzelne kleine Bestandteile isoliert unter die Kontrolle von Vue.js bringen. Das Framework lässt sich Schritt für Schritt einführen. Auch für das schnelle

Prototyping kann Vue.js gute Dienste leisten, zum Beispiel, wenn man kurzfristig etwas Lauffähiges vorzeigen muss, um rasch Feedback vom zukünftigen Benutzer der Anwendung abholen zu können.

Neben der guten Performance gibt es jenseits der Technik weiche Faktoren, die für dieses Framework sprechen. Die Dokumentation etwa, die online auf der Vue.js-Webseite verfügbar ist, erweist sich als hilfreich. Oft heißt es bei Open-Source-Projekten, sie hätten keine, und wenn, eine schlechte Dokumentation. Das gilt mitnichten für Vue.js. Nicht unerwähnt bleiben soll auch die funktionierende Community. In der Vue.js-Community gehen die Entwickler stets freundlich und verständnisvoll miteinander um. Fragen werden in der Regel schnell, unkompliziert und anständig beantwortet.

Alles zusammengesetzt

Der Begriff Komponente leitet sich vom lateinischen Partizip I des Verbs componere ab und bedeutet zusammensetzend [4]. Häuser sind ein gutes Beispiel für Komponentenorientierung außerhalb der IT. Zuerst muss beispielweise im Rahmen des Tiefbaus ein Loch für die Kellerkomponente gegraben werden, wobei der Keller wieder aus eigenen Komponenten wie Bodenplatte, weißer Wanne und Decke besteht. Auf dem Keller sitzt die Komponente Erdgeschoss, gefolgt von Obergeschoss und Dach. Jede dieser Einzelkomponenten setzt sich wieder aus eigenen Komponenten zusammen. Und für jede dieser Komponenten gibt es Spezialisten, die sie planen, umsetzen und warten. Tiefbauer, Maurer, Putzer, Elektriker, Zimmermänner – jeder ist Spezialist und Meister auf seinem Gebiet.

Einen weiteren guten Vergleich, der recht anschaulich zeigt, wie das Ergebnis aussehen kann, wenn man meint, alles selbst machen zu können, ist in Abbildung 1 zu sehen: Der Versuch eines Studenten des Royal College of Arts in London. Leider wird heutzutage in der Softwareentwicklung das Thema Komponentenorientierung immer noch zu selten ernst genommen. So meinen leider viele Entwickler, besonders kreativ sein zu müssen, und

unterstützen uns moderne Webanwendungsframeworks wie React, Angular oder Vue.js an dieser Stelle nicht nur, komponentenorientiert zu denken, sondern sie zwingen uns dazu.

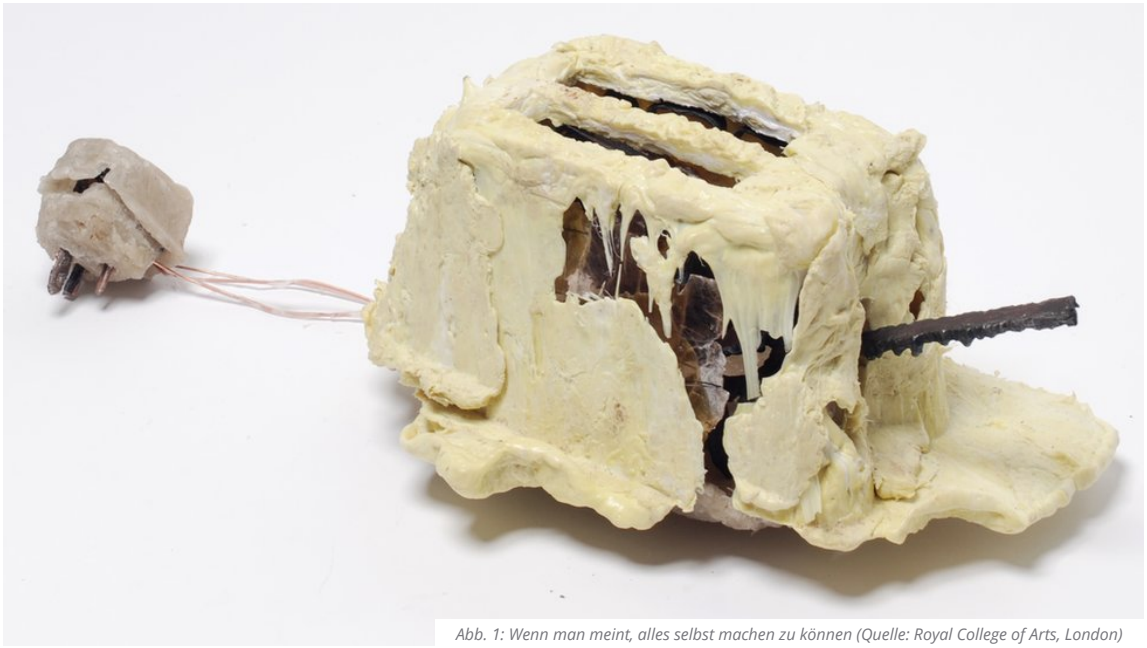


Abb. 1: Wenn man meint, alles selbst machen zu können (Quelle: Royal College of Arts, London)

Komponenten in Vue.js: Wiederverwendung nicht um jeden Preis

Das Komponentenkonzept von Vue.js zielt darauf ab, einfach pflegbare und wiederverwendbare Einheiten von Code zu erstellen. Wobei mit Wiederverwendbarkeit nicht Wiederverwendung um jeden Preis gemeint ist. Es geht vielmehr darum, eine fachliche Anforderung, zum Beispiel eine Anwendung zum Erfassen von Kleinanzeigen, in kleine Bestandteile aufzuteilen. Ziel ist, diese kleinen Komponentenbausteine besser überblicken zu können. Die Anwendung wird in eine hierarchische Struktur von Softwarebausteinen aufgeteilt, aus denen sie sich Stück für Stück

Lesen Sie auch: Vue.js unter der Lupe: Kommunikation zwischen Komponenten

Bei Bedarf einen dieser Bausteine in einem anderen Kontext wiederzuverwenden, ist natürlich sinnvoll. Es sollte aber nicht der Anspruch des Entwicklers sein, Wiederverwendung und generische Routinen um jeden Preis zu entwickeln. Gerade hier lauert nämlich die Gefahr, dass man sich selbst Probleme schafft und unbewusst Komplexität ins Projekt holt. Und ganz ehrlich, es ist auch kein Beinbruch, wenn Code an der einen oder anderen Stelle mal redundant gehalten wird. Beim Hausbau kommt ja auch nicht immer die gleiche Säge zu Einsatz. Der Zimmermann nutzt seine eigene, speziell auf ihn zugeschnittene Säge, kann mit der Säge des

Fliesenlegers jedoch wenig anfangen. Redundanz ist nicht per se schlecht, und wenn sie einem besseren Verständnis der Softwarearchitektur dient, ist sie mit Sicherheit auch kein Fehler.

Die erste Komponente

Betrachtet man eine typische Webanwendung, findet man oben rechts meist eine Anzeige, welcher Benutzer gerade angemeldet ist. Dies lässt sich durch eine Komponente realisieren, ein Baustein, der sich an anderen Stellen der Anwendung wiederverwenden lässt. Das Formular zur Erfassung der Kleinanzeigen, im Kern eine <form>, kann die nächste Komponente bilden. Weitere Bestandteile des Formulars, etwa eine Datumsauswahl, können ebenfalls als Child-Komponenten realisiert werden, die von der Kleinanzeigenerfassungskomponente (der sogenannten Parent-Komponente) verwendet wird.

Eigentlich sind Komponenten gruppiertes HTML. Bestünde eine Komponente jedoch nur aus HTML, wäre das Ganze eine langweilige und statische Angelegenheit. Durch CSS und JavaScript werden Komponenten aufgehübscht und erhalten Dynamik. Frameworks wie React, Angular und natürlich Vue.js bieten uns, ein jedes auf seine ganz bestimmte Weise, die Möglichkeit, alle diese Bestandteile zu Komponenten zu vereinigen. Im Fall von Vue.js ist das durchaus auch wörtlich gemeint. Komponenten bringen die Möglichkeit mit sich, HTML um neue Tags (Custom Tags) zu erweitern. Aus der Log-in-Komponente wird so ein <login>-Tag, und das gesamte Erfassungsformular für Kleinanzeigen lässt sich plötzlich mit einem simplen

Listing 1 zeigt, wie einfach sich Komponenten in Vue.js erstellen lassen. Die Komponente wird über die statische Methode component auf dem Vue.js-Objekt registriert. Grundsätzlich werden Vue.js-Komponenten, wie auch Vue.js-Instanzen, als JavaScript-Array mit Name-Value-Paaren definiert. So wird in Listing 1 die Hello-World-Komponente unter dem Namen hello-world registriert und in der Option template das HTML-Fragment definiert, das ausgegeben werden soll, wenn man den Tag <hello-world> verwendet. Wer sich jetzt fragt, warum die Komponente nicht hello oder helloWorld genannt wurde, der sei an dieser Stelle auf den offiziellen Vue.js-Styleguide verwiesen [5]. Da es keine wirkliche Konvention gibt, wie man Custom-

HTML-Tags benennen soll, werden hier wertvolle Tipps gegeben. So sollte man Einzelwörter vermeiden, um potenzielle Namenskonflikte zu verhindern. Abgesehen davon, dass es relativ unwahrscheinlich ist, dass es im HTML-Standard jemals einen Tag <hello> geben wird, geht die Wahrscheinlichkeit, dass es ein <hello-world>-Tag geben wird, gegen null. Obwohl laut Styleguide das Verwenden von Camel Case durchaus erlaubt ist, hat sich in der Praxis gezeigt, dass ein Camel-Case-artiger Komponentenname wie etwa HelloWorld zum Beispiel im Chrome-Browser nicht auf Anhieb funktioniert und als helloworld interpretiert wird. Dies führt zu Fehlern. Deshalb ist an dieser Stelle Multi Wording, getrennt durch Bindestrich, empfohlen.

Listing 1: Die erste Vue.js-Komponente

```
<html>
  <head>
    <script src="https://unpkg.com/vue"></script>
  </head>
  <body>
    <div id="app"> <hello-world /> </div>
    <script>
      Vue.component('hello-world', {
        template: '<div>Hello World from Vue-Componer
      });
      new Vue ({ el: "#app" });
    </script>
  </body>
</html>
```

Global registrierte Vue.js-Komponenten wie in Listing 1 können überall in der Anwendung zum Einsatz kommen, auch wenn einzelne Bestandteile der

Selbstverständlich lassen sich Komponenten auch an einzelne Vue.js-Instanzen binden, hierzu meldet man die Komponente über die Option components bei der gewünschten Vue.js-Instanz an. Listing 2 zeigt ein Beispiel. Sicher ist die im Listing dargestellte Lösung, die Komponente vorab als Konstante zu definieren, der sinnvollere Ansatz, um die Komponenten später bei Bedarf auch bei anderen Vue.js-Instanzen registrieren zu können. Allerdings ist es denkbar, die Registrierung direkt an der Option components vorzunehmen.

Es ist wichtig zu verstehen, dass Komponenten wiederverwendbare Vue.js-Instanzen sind. Daher teilen sich Vue.js-Komponenten viele Eigenschaften mit Vue.js-Instanzen. Optionen wie data, computed, watch oder auch

methods können bei Komponenten genauso verwendet werden, wie man es von den normalen Vue.js-Instanzen gewohnt ist. Das Event-Modell existiert analog für Komponenten. Speziellere Optionen wie el, um eine Vue.js-Instanz an ein bestimmtes HTML-Tag zu binden, gibt es bei Komponenten nicht. Das verwundert nicht weiter, da Komponenten selbst zu HTML-Tags werden. Die Nummernauswahlkomponente wird in Listing 2 bewusst mehrfach verwendet, und der aufmerksame Leser wird schnell erkennen, dass hier jede Komponenteninstanz in der Lage ist, ihren Zustand selbst zu verwalten. Möglich wird das dadurch, dass an der Option data – im Gegensatz zu Vue.js-Instanzen – anstelle der Daten eine Funktion definiert wird. Rein theoretisch wäre es denkbar, dass ein Entwickler statt dieses (richtigen) Codes

```
data: function() {
  return {
    count: 0
  }
}
```

folgende vereinfachte (falsche) Variante schreibt:

```
data: {
  count: 0
}
```

Ungeachtet dessen, dass Vue.js die vereinfachte Variante gar nicht akzeptieren und ausführen würde, sollte einleuchten, dass dies bei potenziell mehreren Instanzen der Komponente nicht funktionieren kann.

Listing 2: Eine Komponente direkt an eine Vue.js-Instanz b


```
<head>
  <script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id="app">
    <select-number initial='100' suffix='EUR'> </se
    <select-number> </select-number>
    <select-number initial='200' suffix='SFR'> </se
  </div>
  <script>
    const SelectNumberComponent = ('select-number',
      data: function() {
        return {
          count: this.initial
        }
      },
      // props: ['suffix', 'initial'],
      props: {
        suffix: {
```

```
            type: String,
            required: true
          },
          initial: {
            type: String,
            required: false,
            default: '10'
          }
        },
        template: "<div><input type='number' v-model=
      });
      const my = new Vue ({
        el: "#app",
        components: { 'select-number': SelectNumberC
      });
    </script>
  </body>
</html>
```

Von oben nach unten

Ein wichtiges Konzept von Vue.js-Komponenten ist ihr Alleinstellungsmerkmal. Komponenten sollen für sich stehen können und möglichst keine Verflechtungen und Beziehungen zu anderen höher liegenden Komponenten unterhalten. Wer dieses Konzept durchhält, erhält am Ende das bereits erwähnte Baukastensystem, aus dem sich die Anwendung einfach zusammenstecken lässt. Mehr noch, man wird mit kleinen, gut überblickbaren Softwareblöcken belohnt, die besser zu pflegen sind. Das Thema Testbarkeit soll an dieser Stelle nicht unerwähnt bleiben, denn kleine, in sich geschlossene und isolierte Softwareeinheiten lassen sich viel einfacher (und schneller) mit Unit-Tests abdecken.

nur nach unten weitergegeben werden dürfen. Verflechtung untereinander sollte so nicht vorkommen, denn das Programmiermodell von Vue.js sieht nur einen Weg vor: von oben nach unten. Um Daten nach unten weiterzugeben, wird die Option props angeboten, was für Properties steht. Hier kann entweder ein Array von String-Objekten übergeben werden oder eine Auflistung von Objekten. Listing 2 zeigt ein Beispiel. Das auskommentierte props demonstriert die Variante mit String-Array, die Werte hinter initial und suffix – die eigentlichen Daten werden in Form von Eigenschaften am Komponenten-Tag übergeben – kommen in der Komponente an. Die zweite props-Variante erlaubt explizitere Angaben. In Vue.js nennt sich dieses Feature übrigens Props Validation. So kann unter

anderem festgelegt werden, welcher Datentyp erwartet wird (String, Number, Boolean, Functions, Objects), und ob eine Property verpflichtet übergeben werden muss (required). Auch Defaultwerte lassen sich hier festlegen.

Lesen Sie auch: Vue und VueX: Anwendungen im Team skalieren

Mit der zweiten <select-number>-Komponente in Listing 2 wird weder ein Initial noch ein Suffixwert übergeben. Der Zähler beginnt in dieser Komponenteninstanz mit dem Wert 10. Allerdings wird in der Konsole ein Fehler ausgegeben, der darauf hinweist, dass es sich bei Suffix um ein Pflichtfeld handelt, für das kein Wert übergeben wurde.

Der typische Java-Entwickler wird sich jetzt fragen, wie er ganz explizit auf die Daten oder den Zustand einer bestimmten Komponenteninstanz zugreifen kann, wenn Daten nur von oben nach unten durchgereicht werden dürfen. Normalerweise würde man jetzt durch die Liste der vorhandenen Komponenten iterieren und einen Getter aufrufen.

In Vue.js soll man genau das nicht tun. Daten von Parent nach Child weiterreichen heißt in diesem Zusammenhang einfach, dass die Daten in der obersten, wenn nicht sogar in der Wurzelkomponente, gehalten und verwaltet werden. Einzelne Datenbestandteile werden nach unten durchgedrückt, Veränderungen an den Datensätzen bleiben somit automatisch in der Datenquelle ganz oben.

Custom Events und Referenzen

Für den hoffentlich seltenen Fall, dass eine Child-Komponente Information nach oben weitergeben muss, kann man auf ein spezielles Event Handling zurückgreifen: die Custom Events. Hier löst die Child-Komponente ein Event aus und benachrichtigt damit ihren Parent. Hierzu muss die Parent-Komponente mithilfe der v-on-Direktive an ein Event gebunden werden. Außerdem muss festgelegt werden, welche JavaScript-Funktion aufzurufen ist, wenn dieses Event eintritt. In der Child-Komponente hat man dann die Möglichkeit, mithilfe der von Vue.js bereitgestellten Funktion $emit ein Event auszulösen. Hier sind Event-Name und Daten zu übergeben. Die übergeordnete Komponente lauscht auf dieses Event, wird benachrichtigt und erhält die Daten. Dieses Prinzip, von oben nach unten durchzureichen,

gehört zum guten Stil in Vue.js-Anwendungen und sollte nicht durchbrochen werden.

Für die hoffentlich noch selteneren Fälle, in denen es nicht anders geht und man auf Interna von Child-Komponenten zugreifen muss, bietet Vue.js auch die Möglichkeit, mit Referenzen zu arbeiten. Das sieht in der Praxis dann so aus, dass man an jeder Komponente, in die man greifen möchte, die Eigenschaft ref setzt und die Komponente mit einem eindeutigen Namen versieht. In der übergeordneten Komponente hat man mit der Funktion $refs die Chance, eine Auflistung aller mit ref markierten Komponenten zu erhalten. Auf diesem Weg kann man sehr bequem in einzelne Komponenten greifen.

Single File Components

Es ist den Entwicklern von Vue.js wichtig, den Quellcode einer Anwendung so in Form zu bringen, dass er auch in Zukunft ohne große Hürden weiterentwickelt werden kann. Komponenten sind ein wichtiges Hilfsmittel hierfür. Werden alle Komponenten, wie in Listing 2, direkt in eine HTML-Datei oder auf mehrere JavaScript-Dateien verteilt, ist der Quellcode womöglich noch zu sehr verdichtet. Wer später an Komponente B etwas ändern möchte, wird trotzdem mit dem Quellcode der Komponenten A und C belästigt, was zu unnötigen Kontext-Switchen führen und den Entwickler ablenken kann.

Um auch auf Dateiebene einen klaren Schnitt zu machen, ließe sich jede

zweifelsohne funktionieren, jedoch bleibt das Problem, dass dann nur reines JavaScript verwendet werden kann. Bestandteile aus HTML oder CSS müssen in Strings verpackt werden.

Moderne IDEs wie Visual Studio Code, Atom oder Webstorm bringen aber allesamt Syntax Highlighting und Codevervollständigung mit. Das möchte man als Entwickler natürlich nutzen. Reacts Lösung für dieses Problem heißt JSX. Die Idee von JSX (JavaScript eXtended) lautet, HTML und JavaScript mischen zu können. Moderne Build-Systeme wie webpack [6] oder Browserify helfen, aus diesem Spezialformat wieder ausführbares JavaScript zu erzeugen.

Vue.js geht mit seinen Single File Components in eine ähnliche Richtung, allerdings basierend auf einem ganz anderen Konzept. Der Grundgedanke ist, sämtliche Bestandteile der Komponente in einer Datei möglichst gut lesbar zu halten. Die einzelnen Bestandteile werden daher durch spezielle Tags voneinander abgegrenzt. Der HTML-Teil wird innerhalb zweier <template>-Tags abgelegt, der JavaScript-Code befindet sich im Bereich <script>, und für CSS-Anteile ist der Bereich <style> vorgesehen. Abgespeichert wird die Datei schließlich mit der speziellen Dateiendung .vue. Die meisten IDEs können mittlerweile mit Vue.js-Dateien umgehen. Selbstverständlich sind Single File Components nicht ohne weiteres lauffähig, auch diese müssen mithilfe eines Build-Systems in Form gebracht werden. Das bedeutet, JavaScript wird kompiliert, oder besser ausgedrückt transpiliert. Glücklicherweise bietet Vue.js CLI [7] ein entsprechendes Template, mit dem webpack bereits vollständig konfiguriert wird.

Vue.js Tutorial – Fazit

Mit Vue.js lassen sich nach relativ kurzer Eingewöhnungsphase gut wartbare, auf Komponenten basierende Anwendungen bauen. Besonderes Augenmerk legt Vue.js darauf, dem Entwickler Hilfsmittel an die Hand zu geben, um den Quellcode möglichst gut zu strukturieren. Selbstverständlich konnte diese Artikelserie Vue.js nicht in der Tiefe betrachten. Wir hoffen, dass wir Neugierde auf dieses interessante, aufstrebende Framework wecken konnten. Und bitte nicht vergessen: Beschäftigt euch mit JavaScript und denkt in Komponenten! We're all living in a component-oriented world,

Links & Literatur

[1] Vue.js: https://vuejs.org/

[2] Website von Mocha: https://mochajs.org/

[3] Website von Jest: https://facebook.github.io/jest/

[4] Begriffserklärung Komponente: https://www.duden.de/rechtschreibung/Komponente

[5] Der offizielle Vue.js-Styleguide: https://vuejs.org/v2/style-guide/

[6] Website von webpack: https://webpack.js.org/

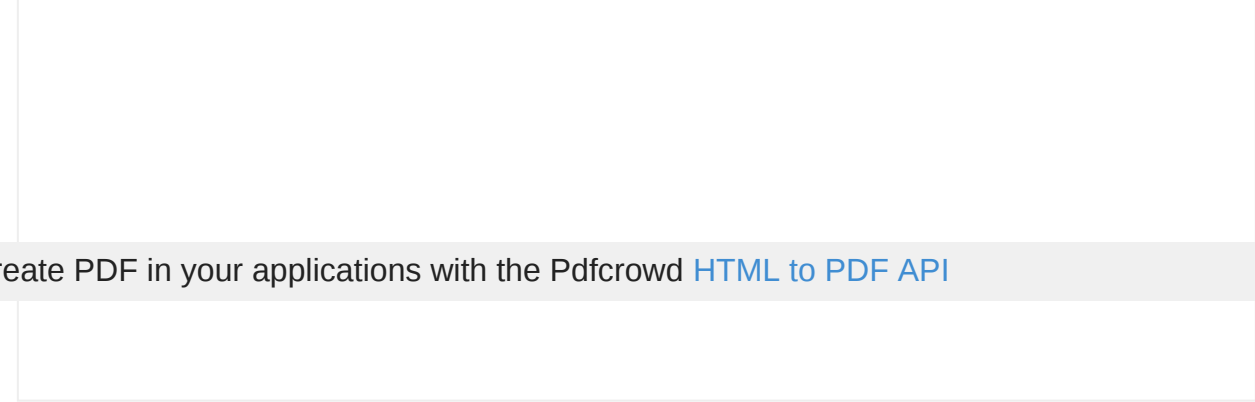
[7] Vue.js-CLI-Kommandozeilenwerkzeug: https://github.com/vuejs/vue-cli

JAVASCRIPT TUTORIAL VUE VUE.JS

RELEVANTE BEITRÄGE

Progressive Web Apps ohne Framework: Geht das?

Von **Redaktion** / 2 Tagen online



Vue.js, ECMAScript und ActiveJ – Unsere Highlights der letzten Woche

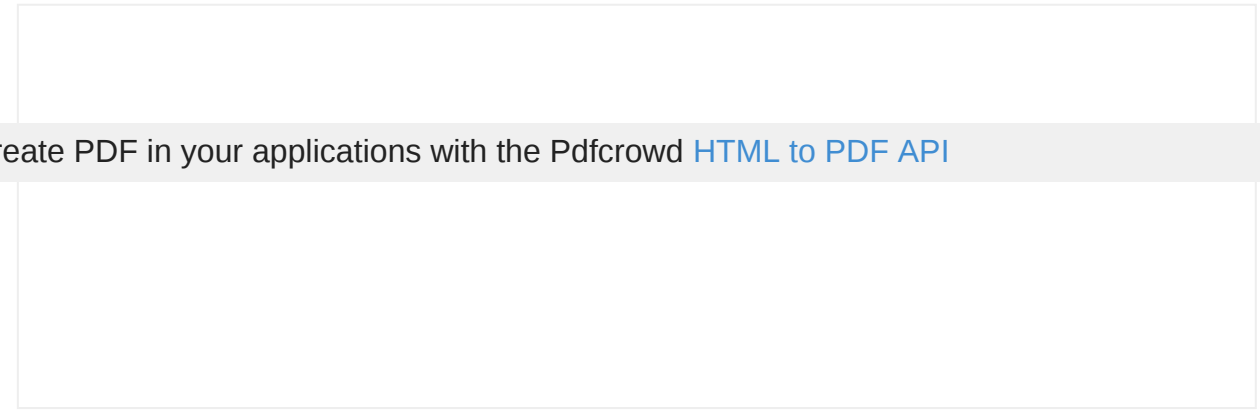
Von **Jean Kiltz** / 3 Tagen online

International JavaScript Conference London – The Online Conference: Bis 30. Juli anmelden und sparen

Von **Redaktion** / 3 Tagen online

TypeScript-Tipps für React-Entwickler

Von **Redaktion** / 1 Woche online



ECMAScript 2021: Vier weitere Features stehen fest

Von **Ann-Cathrin Klose** / 1 Woche online

Vue.js 3: Release Candidate veröffentlicht

Von **Ann-Cathrin Klose** / 1 Woche online

Hinterlasse einen Kommentar

2 Kommentare auf "Vue.js Tutorial: So entwickelt man Komponenten mit Vue.js"

Beteilige dich an der Dikussion

Subscribe

Gast

hyoga

immer wieder gute Vue Artikel die man hier findet. vielleicht mal einen Vue + .net Core = SPA <3 Artikel?

Antworten

2 Jahre her

Editor

Ann-Cathrin Klose

Hallo Hyoga,

danke für das Feedback – es freut uns, dass die Artikel gut ankommen! Bezüglich Deines Vorschlags werden wir uns mal umhören, ob wir einen Autor dafür finden. Falls Du selbst Interesse hast, für unsere

Website oder das Entwickler Magazin darüber zu schreiben oder jemanden kennst, kannst Du Dich gern unter redaktion@entwickler-magazin.de bei uns melden.

Viele Grüße,
Ann-Cathrin

Antworten

2 Jahre her

BASTA!

International PHP Conference

International JavaScript Conference

webinale

MobileTech Conference

Internet of Things Conference

DevOpsCon

API Conference

Machine Learning Conference

SERVICE

Leserservice & Kontakt

Anzeigenservice & Werbung

Autor werden

Digital lesen

Social Media

Entwickler Akademie

Über S&S Media

AGB Lieferkonditionen