Web-App-Zauberkasten

Dynamische Webanwendungen mit Vue.js

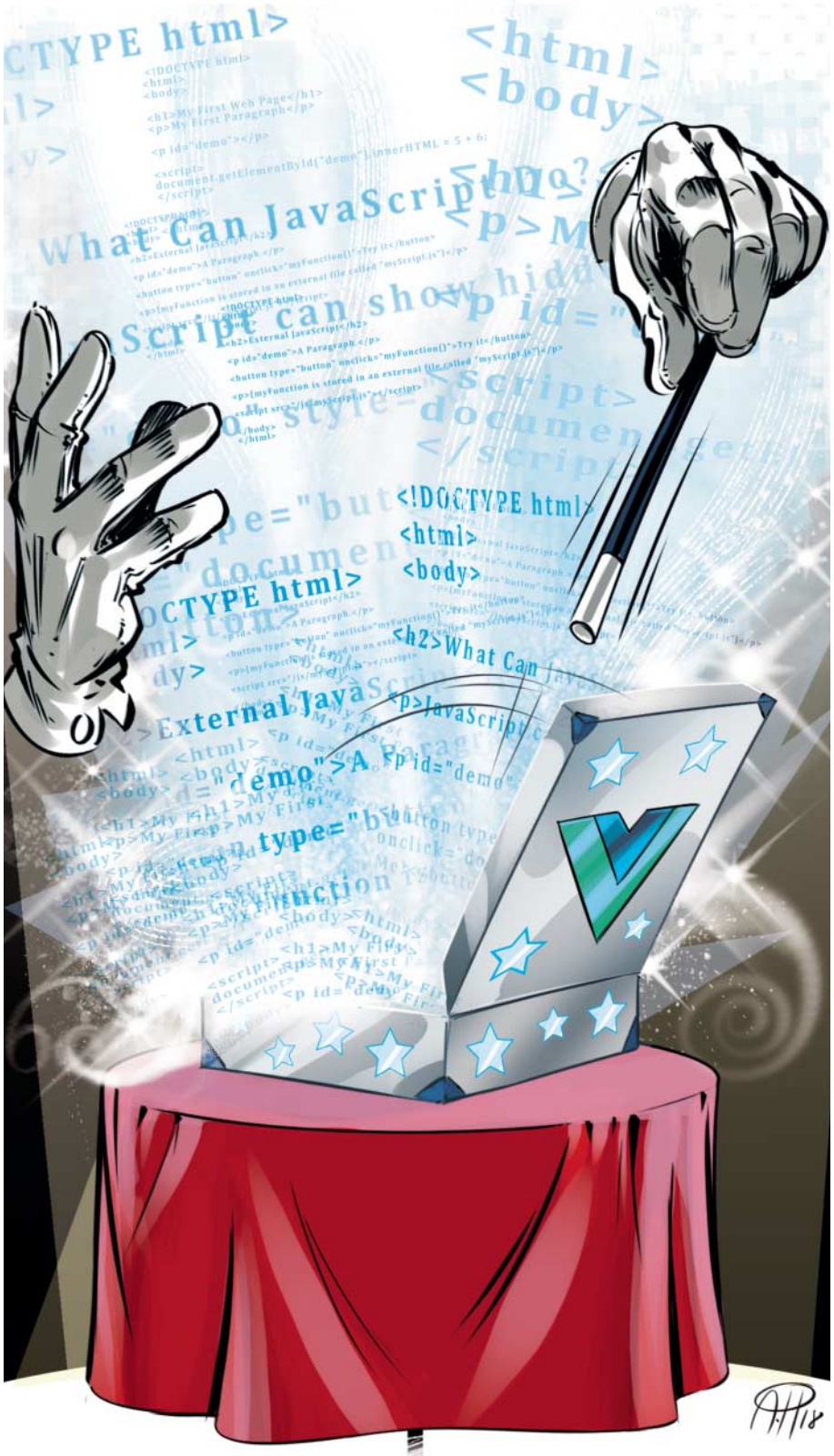


Bild: Albert Hulm, Illustrator

Das JavaScript-Framework Vue.js hat sich ohne die Unterstützung eines großen Unternehmens zu einem ernsthaften Konkurrenten für Angular und React gemausert. Das liegt auch an der gleichmäßigen Lernkurve für Vue-Einsteiger: Nach dem problemlosen Start kann sich der angehende Vue-Entwickler allmählich hocharbeiten – wie unser Praxisbeispiel zeigt.

Von Herbert Braun

Es gibt viele Gründe, die Vue.js innerhalb weniger Jahre zu einem der beliebtesten Frontend-Frameworks gemacht haben – in einem Feld, in dem es wahrlich starke Konkurrenten gibt. Vue-Erfinder Evan You beschreibt das Erfolgsrezept des Frameworks folgendermaßen: „Wie wäre es, wenn ich den Teil con Angular nehme, der mir wirklich gefällt, und etwas Leichtgewichtiges ohne die ganzen Extra-Konzepte drumherum baue?“ Um Vue.js kennenzulernen, haben wir unser mit React entwickeltes Beispiel aus [1] mit Vue.js neu umgesetzt: ein HTML-Quiz, bei dem der Spieler innerhalb einer vorgegebenen Zeit möglichst viele HTML-Elemente eintippen soll.

Erstflug

Die ersten Gehversuche beginnen mit einem leeren HTML-Rahmen:

```
<!doctype html>
<html lang="de">
 <head>
  <meta charset="utf-8">
   <title>HTML-Spiel</title>
   <link rel="stylesheet"
    href="htmlgame.css">
 </head>
 <body>
  <noscript>JavaScript benötigt
   </noscript>
```

```
  <main></main>
   <script src="vue.js"></script>
   <script src="htmlgame.js">
   </script>
  </body>
</html>
```

Das `<main>`-Element ist der Platzhalter für die Anwendung selbst. Das CSS spielt für das Projekt keine Rolle; im Repository unter ct.de/yhet können Sie ein Stylesheet und alle übrigen Projektdaten herunterladen. Vue.js können Sie von https://vuejs.org/js/vue.js (oder /vue.min.js für die minifizierte Version) beziehen. Alternativ lässt sich Vue.js aus dem CDN direkt einbinden. Bei Bower und npm heißt es jeweils schlicht „vue“. So oder so sollten Sie nun in der Skriptdatei htmlgame.js auf das Objekt `Vue` zugreifen können.

Am einfachsten lässt sich Vue mit HTML-Templates nutzen. Für ein „Hallo Welt“ setzen Sie in das `<main>`-Element folgende Zeile ein:

```
<p>{{hallowelt}}</p>
```

Das passende Skript dazu lautet:

```
const test = new Vue({
 el: 'main',
 data: {
  hallowelt: 'Hallo, Welt!'
 }
})
```

Sie erzeugen also eine Vue-Instanz, der Sie ein HTML-Element `el` als Geltungsbereich zuweisen. Dieses beschreiben Sie durch einen CSS-Selektor, der in diesem Fall das `<main>`-Element referenziert (Sie könnten zum Beispiel auch `'#id'` oder `'.class'` nehmen). Wenn es mehrere passende Elemente gäbe, wählt Vue das erste.

Die `data`-Eigenschaft listet Variablen auf, die Vue bei Änderungen automatisch aktualisiert. `hallowelt` enthält einen simplen String, zulässig wären auch andere JavaScript-Datentypen. Im Template gibt es je nach Einsatzzweck unterschiedliche Wege, die Vue-Variablen zu referenzieren; bei Textinhalten stehen sie in doppelten geschweiften Klammern.

Wenn Sie die HTML-Datei im Browser öffnen, hat Vue den Variablenwert ins Markup eingesetzt – auch mehrfach, wenn Sie `hallowelt` innerhalb von `<main>` öfter referenzieren. Öffnen Sie nun die Entwicklerwerkzeuge des Browsers und geben Sie in der Konsole Folgendes ein:

```
test.hallowelt = 'Hallo, Vue!'
```

Daraufhin aktualisiert sich die Seite. Vue hat dem Vue-Objekt `test` als Eigenschaft `hallowelt` zugewiesen umd setzt Änderungen ihres Wertes sofort in HTML um. In der Vue-Terminologie ist diese Variable „reaktiv“.

Nach dem gleichen Muster bauen Sie auch komplexere Anwendungen. In der Template-Sprache legen Sie HTML-Attribute, Events, Bedingungen, Schleifen und bidirektionale Datenbindungen (Two-way binding) fest; das Skript verwaltet Variablen sowie die dazugehörigen Methoden.

Spielprojekt

Dieses Repertoire werden Sie im Beispielprojekt genauer kennenlernen. Dabei müssen diverse Komponenten Werte aktualisieren und miteinander Informationen austauschen.

Im Mittelpunkt der Anwendung steht ein Eingabefeld. Entspricht der dort eingegebene Text dem Namen eines bislang noch nicht eingetippten HTML-Elements, fügt das Skript es der Trefferliste hinzu. Ein Zähler und eine Liste der bereits gefundenen Elemente informieren über den aktuellen Spielstand. Mit der ersten Eingabe startet ein Timer mit einer vorgegebenen Zeit. Nach dem Spielende zeigt das Skript die gefundenen und die übersehenen Elemente an und errechnet einen Score.

Das HTML-Template (wiederum innerhalb von `<main>`) des Eingabefeldes sieht folgendermaßen aus:

```
<input type="text" autofocus
 v-bind:disabled="inputDisabled"
 v-model:value="inputValue"
 v-on:input="handleInput">
```

Der passende Skript-Code ist nicht viel komplizierter als im vorigen Beispiel:

```
const game = new Vue({
 el: 'main',
 data: {
  inputValue: '',
  inputDisabled: false
 },
 methods: {
  handleInput: function() {
   console.log(this.inputValue);
  }
 }
})
```

Falls Sie im Template vergeblich die doppelten geschweiften Klammern aus dem letzten Beispiel gesucht haben: Innerhalb eines Elements referenzieren Sie die Vue-Variablen stattdessen über spezielle Attribute, die mit `v-` beginnen. `v-bind` holt einen Attributwert ins Element, in diesem Beispiel (`v-bind:disabled`) für das `disabled`-Attribut. Dieses erwartet in `inputDisabled` einen Boole-Wert, der festlegt, ob das Eingabefeld benutzbar ist oder nicht.

`v-bind` hat mit dem obigen `{{hallowelt}}`-Beispiel die Einweg-Bindung gemeinsam: Ein neuer Wert der App-Variablen landet sofort im Markup. Aber sollte sich der Wert dort zum Beispiel durch eine DOM-Manipulation ändern, bekommt Vue nichts davon mit. Eine Zweiwege-Bindung bringt erst `v-model` ins Spiel: Aktualisieren Sie `game.inputValue`, ändert sich der Wert im Eingabefeld; tippen Sie dort etwas hinein, betrifft das wiederum die App-Variable.

Mit `v-on` fangen Sie Events ab – hier das `input`-Ereignis. Den zuständigen Handler `handleInput` legen Sie unter `methods` an. Dafür sollten Sie übrigens nicht die moderne Pfeil-Syntax (`handleInput: () => {…}`) verwenden, weil Sie häufig `this` verwenden werden, um aufs App-Objekt zu verweisen. Auf diese Weise kommen Sie zum Beispiel in einer Methode an die in `data` aufgelisteten Variablen heran. Die Dummy-Funktion im Listing schreibt erst einmal nur den aktuellen Stand der Eingabe in die Konsole.

Die häufig benötigten Attribute `v-bind` und `v-on` lassen sich abkürzen:

```
<input :disabled="inputDisabled"
 @input="handleInput">
```

Statt `v-bind:attribut` genügt `:attribut`, `v-on:event` lässt sich durch `@event` ersetzen.

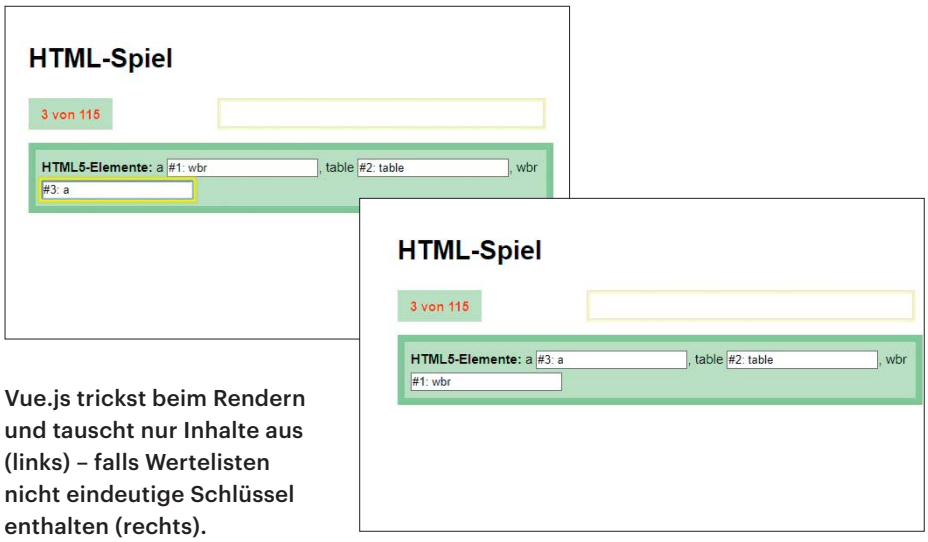
Spieldaten

Um weiterzumachen, benötigen Sie die Spieldaten – also die Daten mit den Informationen zu den HTML-Elementen. Kopieren Sie die 8 KByte große Datei gamedata.js aus unserem GitHub-Repository und binden Sie sie in Ihr Projekt vor htmlgame.js ein:

```
<script src="gamedata.js"></script>
```

Um die Ergebnisse schön aufzubereiten und zusätzlich experimentelle und veraltete Elemente hereinzuholen, enthält die Datei mehr als nur eine Liste der HTML-Elementnamen:

```
const gamedata = [
 {
  name: 'Metadaten',
  elements: [
   {name: 'head',
```



Vue.js trickst beim Rendern und tauscht nur Inhalte aus (links) – falls Wertelisten nicht eindeutige Schlüssel enthalten (rechts).

```
    role: 'metadata container'},
   {name: 'title',
    role: 'document title'},
   /*...*/
  ]
 }, /*...*/, {
  name: 'Veraltet',
  value: 'malus',
  elements: [
   {name: 'font',
    role: 'use CSS instead'},
   /*...*/
 }, {
  name: 'Experimentell',
  value: 'bonus',
  elements: [ /*...*/]
 }
]
```

`gamedata` ist also ein Array von Objekten, die verschiedene Gruppen wie „Metadaten“, „Abschnitte“, „Tabellen“, „eingebettete Inhalte“ als `name` haben; für „Veraltet“ und „Experimentell“ ist zusätzlich noch ein `value` definiert. `elements` ist wiederum ein Array, das die einzelnen Elemente mit `name` und `role` beschreibt.

Um rasch nachzuschlagen, ob die Eingabe einem noch nicht gefundenen Elementnamen entspricht, empfiehlt sich allerdings eine einfachere Struktur – zum Beispiel je ein Set für veraltete, experimentelle und Standardelemente. Sets sind unsortierte Kollektionen von Werten, die keine Doppelungen akzeptieren:

```
const els = {}
const groups = ['html5',
 'experimental', 'deprecated'];
groups.forEach(group =>
 els[group] = new Set());
gamedata.forEach(group => {
 if (group.value === 'bonus')
  group.elements.map(el =>
   els.experimental.add(el.name));
 else if (group.value === 'malus')
  group.elements.map(el =>
   els.deprecated.add(el.name));
 else
  group.elements.map(el =>
   els.html5.add(el.name));
});
```

Das Objekt `els` enthält nun die drei Sets `html5`, `experimental` und `deprecated`. Jedes von ihnen hat eine Reihe von HTML-Elementnamen als Werte.

Schlüsselsuche

Damit lässt sich der Kern des Spielprinzips umsetzen. Ergänzen Sie dazu als Erstes drei leere Arrays in `data`, die genauso heißen wie die drei Gruppen:

```
data: {
 /*...*/
 html5: [],
 experimental: [],
 deprecated: []
}
```

In diesem Fall brauchen Sie Arrays, da die Ausgabe sortiert erfolgen soll. In der `handleInput()`-Methode überprüfen Sie nun, ob sich die Eingabe mit einem bekannten Element deckt:

```
handleInput: function() {
 groups.some(group => {
  if (els[group].
   has(this.inputValue)) {
    if (this[group].indexOf(this.
     inputValue) < 0) {
      this[group] = this[group].
       concat(this.inputValue).sort();
      this.inputValue = '';
    }
   return true;
  }
 });
}
```

Die Methode `some()` durchläuft (ähnlich wie das bekanntere `forEach()`) das `groups`-Array, bricht aber ab, sobald die Callback-Funktion `true` zurückgibt. Das Callback ist bewusst als Pfeil-Funktion geschrieben, damit es das `this` der Elternfunktion übernimmt. Das äußere `if` prüft, ob der Inhalt der Eingabe `this.inputValue` in der jeweiligen Map `els[group]` vorhanden ist. Falls ja, bricht `return true` die weiteren Suchdurchläufe ab.

Zuvor sucht aber das innere `if` im passenden `data`-Array `this[group]` nach dem betreffenden Element. Existiert es dort nicht (`indexOf(…) < 0`), fügt `concat()` es dem Array hinzu, das gleich für die spätere Ausgabe neu sortiert wird (`sort()`). Die nächste Zeile setzt das Eingabefeld zurück.

Wenn Sie nun ein neues HTML-Element eintippen, leert sich das Eingabefeld. In der Konsole oder mit einer der Vue-Entwickler-Erweiterungen für Chrome und Firefox können Sie verfolgen, wie `game.html5` und die anderen beiden Arrays allmählich wachsen.

Zähler und Ausgabe

Um das zu visualisieren, können Sie mit einfachen Mitteln einen Zähler in die Oberfläche einbauen:

```
<div id="counter">{{done}} von
 {{todo}}</div>
```

Ergänzen Sie nun die beiden Variablen in `data`:

```
data: {
 /*...*/
 todo: els.html5.size,
 done: this.html5.length
}
```

`todo` enthält die Gesamtzahl der HTML5-Elemente im Set, `done` bezieht sich auf das Vue-Array `html5` mit den bereits gefundenen Elementen. Wenn Sie die Seite nun neu laden, sehen Sie … einen JavaScript-Fehler: Vue kann noch nicht auf `this.html5` zugreifen, wenn es die Variablen einliest. Die Lösung dafür ist die `computed`-Eigenschaft, die neben `data` und `methods` auf der obersten Ebene des Vue-Objekts steht – damit können Sie aus reaktiven Variablen neue berechnen:

```
const game = new Vue({
 data: {...},
 computed: {
  done: function() {
   return this.html5.length
  }
 },
 methods: {...}
})
```

Anders als `data` erwartet `computed` den Rückgabewert einer Funktion. Mit jeder Änderung an `game.html5` aktualisiert Vue `game.done`.

Zusätzlich soll das Spiel die bereits gefundenen Elemente auflisten. Die minimalistische Lösung wäre, die Array-Variablen `html5`, `experimental` und `deprecated` in doppelte geschweifte Klammern zu verpacken und einfach ins Template zu schreiben. Das klappt tatsächlich, sieht aber nicht schön aus: `["a", "b", "br"]` …

Besser gelingt es mit dem Vue-Attribut `v-for`, das über ein Array oder Objekt iteriert:

```
<span v-for="el in html5">{{el}}
 </span>
```

Vue holt jedes Element `el` aus dem Array `game.html5`, setzt dafür ein neues `<span>` ein und schreibt das Element als Textinhalt hinein. Beim Iterieren von Arrays kann `v-for` auch die Indexnummer extrahieren (`v-for="(el, i) in html5"`), bei Objekten auf die gleiche Weise den Schlüssel.

Nun aktualisiert Vue bei jedem neu gefundenen Element die Liste. Eine schönere Darstellung mit Leerzeichen und Kommata bewerkstelligt am einfachsten ein kleiner CSS-Kniff, der im Projekt-Stylesheet enthalten ist:

```
span + span::before {
  content: ', ';
}
```

UI-Bibliotheken wie Vue.js tricksen viel beim Rendern, denn DOM-Zugriffe sind zeitaufwendig. Bei Änderungen hängt die Reaktionsschnelligkeit der Oberfläche maßgeblich von der Wiederverwertbarkeit bereits gerenderter Komponenten ab. Wenn Sie Vue dabei über die Schulter schauen wollen, setzen Sie testweise ein `<input>` in den `<span>`:

```
<span v-for="el in html5">{{el}}
 <input></span>
```

Geben Sie nun zuerst einen Elementnamen ein, der weit hinten im Alphabet steht, zum Beispiel „wbr“. Der Zähler springt auf 1, das Element erscheint in der Ausgabeliste, gefolgt von einem Eingabefeld. Tippen Sie dort etwas hinein und schreiben Sie einen weiteren Elementnamen (zum Beispiel „div“) ins Spielfeld. Wie erwartet, sortiert das Spiel „div“ vor „wbr“ – aber das `<input>`-Feld mit Ihrer Eingabe ist nicht mitgewandert und steht jetzt neben „div“. Statt alles neu zu rendern, hat Vue nämlich nur Inhalte ausgetauscht und ans Ende der Liste ein neues Element angehängt.

In manchen Szenarien kann dieses Verhalten zu merkwürdigen Bugs führen. Daher gilt es als gute Praxis, Listenelemente mit einem `key`-Attribut individuell zu markieren (React macht es übrigens ähnlich und setzt bei Nichtbeachtung sogar Warnungen auf der Konsole ab). Das `key`-Attribut sorgt dafür, dass Vue Elemente wiedererkennt und umsortieren kann – was außerdem schneller sein soll als das Default-Verhalten:

```
<span v-for="el in html5"
 :key="'output-' + el">{{el}}</span>
```

Als Schlüsselwert eignet sich hier der Array-Wert selbst am besten, der ja eindeutig ist. Um Verwechslungen zu vermeiden – am Ende des Spiels wird es noch eine Report-Ausgabe geben –, fügen Sie einen String wie `output-` in den Key ein; String-Konkatenation ist im Template kein Problem. Wenn Sie den Test mit dem `<input>`-Feld wiederholen, werden Sie feststellen, dass Vue die Elemente nun tatsächlich umsortiert. Sonst hinterlassen die `key`-Attribute keine Spur im Markup.

Eingekleidet in `<output>`-Elemente sieht das Markup so aus:

```
<output class="green"
 v-if="html5.length > 0">
  HTML5: <span v-for="el in html5"
  :key="'output-' + el">{{el}}</span>
</output>
<output class="blue"
 v-if="experimental.length > 0">
  Experimentell: <span v-for="el in
  experimental" :key="'output-' +
  el">{{el}}</span>
</output>
<output class="red"
 v-if="deprecated.length > 0">
  Veraltet: <span v-for="el in
  deprecated" :key="'output-' +
  el">{{el}}</span>
</output>
```

Die `v-if`-Attribute sind nicht schwer zu verstehen: Die Blöcke erscheinen erst, wenn das betreffende Array Elemente enthält.

Timer und Status

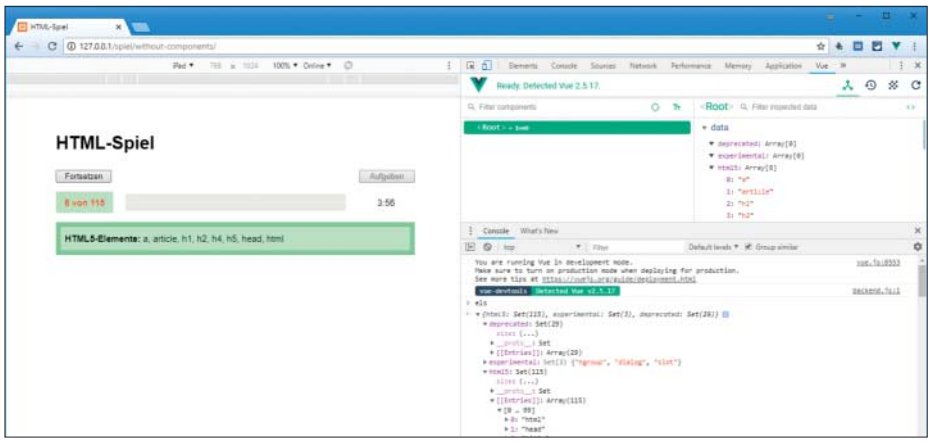
Beim HTML-Elemente-Spiel tickt die Uhr: Mit der ersten Eingabe beginnt der Countdown zu laufen, bis er bei Null angekommen ist oder der Nutzer alle HTML-Elemente gefunden hat. Legen Sie dazu zwei neue Vue-Variablen fest:

```
data: {
 /*...*/
 state: 'off',
 time: 360
}
```

Läuft das Spiel, steht `game.state` auf `'on'`, ist es vorbei, auf `'over'`. `game.time` erfasst die noch verfügbare Zeit in Sekunden; zu Beginn sind es also sechs Minuten. Während des Testens kann es helfen, vorübergehend einen niedrigen Wert einzusetzen. Eine `timer()`-Funktion (siehe Projekt-Code unter dem c’t-Link) zählt die Variable `time` bis auf 0 herunter, dann setzt sie den `state` auf `over`.

Erst einmal müssen Sie sie aber starten. Dazu brauchen Sie eine Anweisung in `handleInput()`. Bei dieser Gelegenheit ergänzen Sie gleich den Statuswechsel zu `'over'`, falls der Nutzer sämtliche Elemente gefunden hat:

Das vollständige Spiel gibt einen ausführlichen Report aus und ermittelt einen Score. Für veraltete Elemente gibts Strafpunkte, für experimentelle einen Bonus.



Mit den Vue-Entwickler-Tools kann man sich live zum Beispiel Variablenbelegungen ansehen – und mit der Konsole schummeln.

```
handleInput: function() {
 if (this.state === 'off')
  this.state = 'on';
 /*...*/
 if (this.done === this.todo)
  this.state = 'over';
}
```

Sie könnten auch gleich die jeweiligen Funktionen in die `if`-Blöcke setzen, aber klarer ist es, den Anwendungs-`state` zu überwachen. Dafür gibt es die `watch`-Option, die auf der gleichen Ebene wie `data`, `methods` und `computed` steht:

```
const game = new Vue({
 /*...*/
 watch: {
  state: function(newState) {
   if (newState === 'on')
    this.timer();
   else if (newState === 'over')
    console.log('Auswertung ...');
  }
 }
})
```

Der Watcher übergibt einer Funktion als Argumente den neuen Wert der gleichnamigen Variable. Damit lässt er sich als App-Controller einsetzen, der den Timer oder bei Spielende die Auswertung startet. Nun legt der Timer mit der ersten Eingabe los. Damit der Spieler auch seine Restspielzeit sehen kann, braucht das Template ein kleines Update:

```
<div id="timer" :class="state">
 {{minutes}}:{{seconds}}</div>
```

Die `minutes` und `seconds` berechnen Sie auf Grundlage von `game.time`:

```
computed: {
 /*...*/
 minutes: function() {
  return Math.floor(this.time / 60);
 },
 seconds: function() {
  const _seconds = this.time % 60
  return _seconds < 10 ? '0' +
    _seconds : _seconds;
 },
 inputDisabled: function() {
  return this.state !== 'off' &&
   this.state !== 'on';
 }
}
```

Ein paar simple Rechenoperationen extrahieren Minuten- und Sekundenwerte, die Vue.js ins Template einfügt und laufend aktualisiert. Bei dieser Gelegenheit können Sie endlich etwas Sinnvolles mit `inputDisabled` anstellen (nachdem Sie es aus `data` entfernt haben): Abhängig vom App-`state` aktiviert und deaktiviert sich das Eingabefeld nun automatisch.

Template-Tricks

Um die React-Version funktional ein klein wenig zu übertreffen, soll die Vue-Version noch einen Pausen- und einen Aufgeben-Button erhalten. Mit der aufeinander aufbauenden Variablen-Aktualisierung und den Template-Fähigkeiten von Vue.js ist das schnell umgesetzt:

```
<button v-if="state === 'over'"
 @click.prevent="restart">
 Neu starten</button>
<button v-else @click.prevent="pause"
 :disabled="pauseDisabled">
 {{state === 'paused'? 'Fortsetzen'
 : 'Pause'}}</button>
<button @click.prevent=
 "state = 'over'"
 :disabled="state !== 'on'">
 Aufgeben</button>
```

`v-if` kennen Sie bereits; unmittelbar danach kann `v-else-if` oder `v-else` folgen. In diesem Fall führt das dazu, dass entweder der Neustart- oder der Pause-Button im DOM auftauchen.

Die drei `@click`-Eigenschaften haben jeweils einen `.prevent`-Zusatz. Dieser ruft die Standardmethode `event.preventDefault()` auf und verhindert so, dass die angeklickten Buttons eine neue URL öffnen.

Einfache Operationen lassen sich auch direkt im Template ausführen – etwa die Beschriftung des Pause-Buttons oder die mit dem Aufgeben-Button verbundene Anweisung. Letztere funktioniert, weil sich der mit der `state`-Variable verbundene Watcher um die Details kümmert. Der Aufgeben-Button lässt sich anklicken, wenn `state` den Wert `'on'` hat. Auch der `pauseDisabled`-Status ließe sich direkt im Template festlegen, aber das wäre etwas sperriger:

```
computed: {
 pauseDisabled: function() {
  return this.state !== 'on' &&
   this.state !== 'paused';
 },
 /*...*/
}
```

Die Pausenfunktion wechselt einfach nur zwischen zwei Anwendungszuständen; `restart()` setzt alle relevanten Variablen zurück:

```
methods: {
 /*...*/,
 pause: function() {
  this.state =
   (this.state === 'paused')?
   'on' : 'paused';
 },
 restart: function() {
  this.inputValue = '';
  this.state = 'off';
  this.time = 360;
  this.html5 = [];
  this.deprecated = [];
  this.experimental = [];
  this.results = [];
 }
}
```

Lebenszyklus

Ähnlich wie seine Konkurrenten verfügt auch Vue über Hooks, die zu bestimmten

Zeiten im App-Lebenszyklus anspringen und mit denen Sie Funktionen auslösen können.

So stört es ein wenig, dass die App nach dem Aufruf einen Augenblick lang im Browser zu sehen ist, bevor Vue sie wieder verbirgt. Die Lösung: Setzen Sie einfach im CSS den App-Selektor `main` auf `display: none` und korrigieren Sie dies, wenn das Spiel bereit ist:

```
const game = new Vue({
 mounted: function() {
  this.$el.style.display = 'block';
 },
 /*...*/
})
```

`this.$el` ist eine Spezialvariable, die sich auf das App-Element bezieht. Noch vor `mounted` löst `created` aus, wo Sie erstmals auf App-Variablen zugreifen können. Weitere Hooks sind `updated`, `destroyed` sowie die jeweilige `before`-Variante, zum Beispiel `beforeMount`.

Komponenten

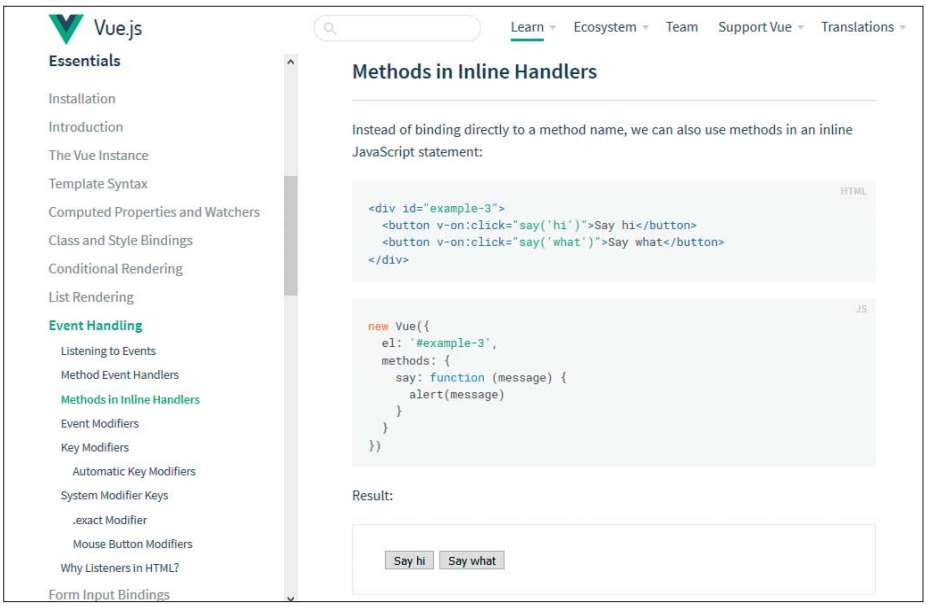
Bei wiederverwendbaren Komponenten setzt Vue.js auf HTML-Custom-Elemente, eine Webtechnik, deren Spezifizierung seit Längerem in Arbeit ist [2]. Custom-Elemente erkennen Sie am Bindestrich im Namen.

Ein einfacher Anwendungsfall ist der Ersatz für die drei `<output>`-Elemente, die während des Spiels den Zwischenstand anzeigen:

```
<found-elements v-if="html5.length"
 :list="html5" color="green">
 HTML5-Elemente</found-elements>
<found-elements v-if="experimental.↵
 ↳length" :list="experimental"
 color="blue">
 Experimentell</found-elements>
<found-elements
 v-if="deprecated.length"
 :list="deprecated" color="red">
 Veraltet </found-elements>
```

Die Elemente übergeben alle notwendigen Werte: die Liste (`list`), die Farbe (`color`) und die Beschriftung als Textinhalt. Die dazugehörige Komponente sieht so aus:

```
const game = new Vue({
 components: {
  'found-elements': {
   props: ['color', 'list'],
   template: `<output :class="color">
    <slot></slot>:
    <span v-for="item in list"
     :key="'output-' + item">
     {{item}}</span>
   </output>`
  }
 },
 /*...*/
})
```



Vue.js überzeugt auch mit seiner guten Dokumentation sowie der sehr regen und hilfreichen Community.

`components` ist eine weitere App-Eigenschaft. Dem Elementnamen als Schlüssel sind die Details der Komponente zugewiesen. Die übergebenen Eigenschaften `color` und `list` müssen Sie noch mal als `props` auflisten. Das Template verwendet Backticks statt Anführungszeichen, was mehrzeilige Strings und Variablen-Interpolierung erlaubt.

Auf den Textinhalt von `<found-elements>` greifen Sie mittels `<slot>` zu – auch das ist keine Erfindung der Vue-Macher, sondern in den jüngsten HTML-Entwürfen für Custom-Elemente bereits so vorgesehen.In der Praxis setzen Webagenturen Komponenten meist anders ein: als global verfügbare, wiederverwendbare Bausteine, die eigene Methoden und sogar Stile mitbringen. Für die globale Registrierung sorgt `Vue.component()`.

Jede Komponente ist typischerweise in einer eigenen Datei mit der Endung .vue untergebracht, wobei der Modul-Bundler Webpack für den Import sorgt. So einen Workflow können Sie sehr einfach über das von Vue.js bereitgestellte Kommandozeilenwerkzeug einrichten (siehe ct.de/yhet).

Vue-Ausblick

Vue kann noch mehr: Es unterstützt animierte Übergänge und erlaubt serverseitige Programmierung. Angular-Fans können Vue-Anwendungen in TypeScript programmieren, React-Liebhaber dürfen die Template-Sprache JSX weiterverwenden. Es gibt Tools fürs Routing, für State-Management im Stil von Flux sowie Browser-Erweiterungen zum Debuggen und das erwähnte Kommandozeilenwerkzeug. Vermutlich ist es mit React- und Angular-Know-how immer noch einfacher, einen Job zu finden, aber sonst spricht viel für das jüngste der drei großen Frameworks. Einen Vorsprung haben die beiden Platzhirsche allerdings noch bei der plattformübergreifenden Entwicklung dank Frameworks wie Ionic, NativeScript oder React Native.

„Die Technik entwickelt sich vom Primitiven über das Komplizierte zum Einfachen“, soll angeblich Antoine de Saint-Exupéry einmal gesagt haben. Mit Angular und React schaffte demnach die Frontend-Entwicklung den Sprung in die komplizierte Phase – aber einfach wird es erst mit Vue.js. *(jo@ct.de)* c't

Literatur

[1] Herbert Braun, Webseiten-Reaktor, Facebooks JavaScript-Bibliothek React für datenlastige Websites, c't 2/2016, S. 162
[2] Herbert Braun, HTML maßgeschneidert, Eigene Elemente und Templates definieren, c't 26/2013, S. 182

Projekt-Download: ct.de/yhet