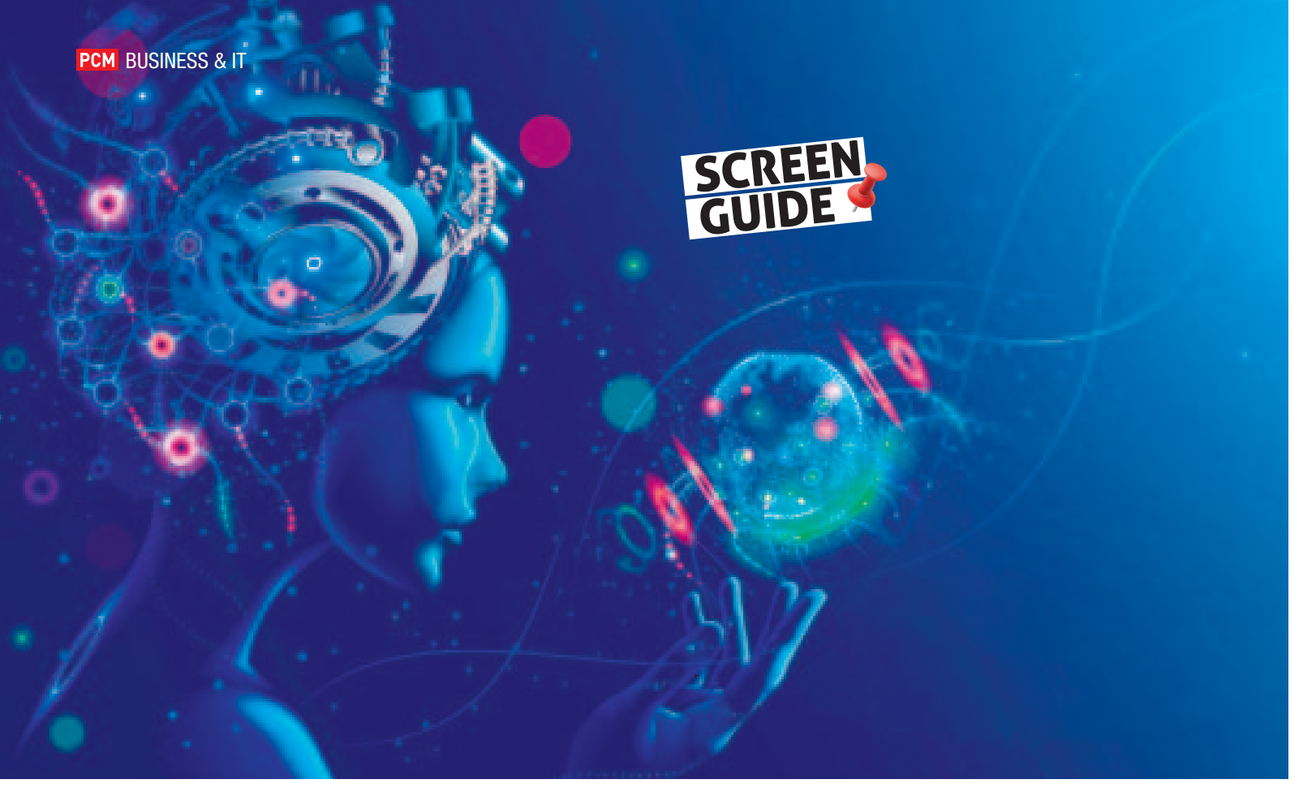

FRONTEND COMPILER

Schnelle Web Apps mit Svelte

Das Framework Svelte wird in den letzten Jahren immer öfter als Alternative zu etablierten Tools wie React und Vue genannt. Dabei punktet Svelte insbesondere mit minimalem Code und einer guten Performance. ■ NICOLAI SCHWARZ

Wenn es um die großen JavaScript UI Frameworks geht, fallen immer wieder Namen wie React, Vue.js, Angular und Ember. In den letzten Jahren wird auch Svelte unter Entwicklern immer beliebter (**svelte.dev**). Mit all diesen Frameworks können Sie Apps oder UI-Komponenten für eine Website bauen – etwa Slider, Lightboxen, Popups oder interaktive Infografiken.

Es gibt daher zwar Gemeinsamkeiten zwischen den Frameworks, aber gerade Svelte handhabt einige Aspekte deutlich anders. So arbeiten React und Vue mit einem *Virtual DOM*, Svelte aktualisiert hingegen direkt das DOM. Bei Frameworks wie React sind der React-Code und weitere Module immer Teil der App. Svelte ist aber in erster Linie ein Compiler. Das bedeutet, dass der gesamte Svelte-Code in optimiertes Vanilla JavaScript kompiliert wird. Das Svelte-Framework selbst wird in der App nicht mehr benötigt. Dieser Ansatz sorgt dafür, dass die generierten Komponenten in Svelte oft kleiner sind und performanter arbeiten. Außerdem ist der Svelte-Code selbst meist übersichtlicher (siehe Infokasten auf der nächsten Seite).

Hello World!

Schauen wir uns ein simples *Hello World* mit einer Variablen *name* in Svelte an:

```
<script>
  let name = "World"
</script>
<h1>Hello {name}!</h1>
```

Bild: Andrey Suslov / Shutterstock.com

Das ist auch für Anfänger einfach zu lesen. Sie müssen nicht groß raten, was diese oder jene Zeile macht. Ein vergleichbarer Code sieht in React etwa so aus (**bit.ly/3518PSi**):

```
import React from 'react';
const Hello = () => {
  let name = "World";
  return <h1>Hello {name}!</h1>;
}
export default Hello;
```

Es gibt auch andere Möglichkeiten, das in React umzusetzen (**bit.ly/34X4eQL**) – aber alle Varianten sind komplizierter als in Svelte. Vor allem wenn Sie die beiden Tutorials auf **svelte.dev** und **reactjs.org** miteinander vergleichen, zeigt sich, dass Svelte insgesamt sehr viel zugänglicher ist.

Svelte im Browser testen

Unter dem Menüpunkt REPL auf **svelte.dev** können Sie den Compiler online ausprobieren. REPL steht für *read*, *evaluate*, *print*, *loop*. Die App liest den aktuellen Code, evaluiert (und kompiliert) ihn und zeigt das Ergebnis an. Das Ganze erfolgt in einem Loop, jede Änderungen wird direkt verarbeitet. Die REPL startet mit dem Hello-World-Code von oben. Sie können hier Ihren eigenen Code ausprobieren. Um den Code forken oder speichern zu können, müssen Sie sich einmal einloggen. Das geht recht schnell, sofern Sie über einen Account bei GitHub verfügen. Für die Beispiele in diesem Artikel reicht es, wenn Sie die REPL nutzen. Sie können mit Svelte aber natürlich auch lokal entwickeln (**bit.ly/2RAPFQ0**).

Eine eigene Komponente

Wir programmieren nun eine eigene, kleine Komponente – eine eigenständige Einheit aus HTML, CSS und JavaScript. Dabei ist die Einheit gekapselt, sodass sich CSS und JavaScript aus verschiedenen Komponenten nicht in die Quere kommen. Hier bauen wir einen div-Container mit einem Button. Bei einem Klick auf den Button soll sich einfach nur die Hintergrundfarbe des div-Containers ändern.

Auch wenn unsere Komponente nur ein paar Zeilen groß wird, lagern wir diese in eine eigene Datei aus. Erzeugen Sie in der REPL über das *Plus*-Icon einen neuen Reiter und nennen ihn *RandomBG.svelte*. Das *RandomBG* ist der Name der Komponente. Die Groß- und Kleinschreibung ist wichtig, *RandomBG* ist also nicht gleich *randombg*. Es ist Konvention, dass eigene Komponenten mit einem Großbuchstaben beginnen, um sie von regulären HTML-Elementen zu unterscheiden, die meist klein geschrieben werden. Wir beginnen mit dem HTML:

```
<div>
  <button>Change Color!</button>
</div>
```

Im Anschluss schreiben wir einen Block mit den nötigen CSS-Anweisungen:

```
<style>
div { display: flex; min-width: 300px;
  min-height: 200px; justify-content:
  center; align-items: center; transi
  tion: background-color 0.5s linear; }
</style>
```

Nun benötigen wir noch ein Script für die zufällige Hintergrundfarbe. Am einfachsten ist es, wenn wir die Farbe im Format *rgb()* angeben und drei Zahlen zwischen 0 und 255 erzeugen. Die Reihenfolge von JavaScript, HTML und CSS ist in der Komponente zwar egal, aber die meisten Beispiele nutzen diese Reihenfolge: Script, Markup, Style. Wir übernehmen diese Konvention. Fügen Sie den folgenden Code also am Beginn der Komponente hinzu:

```
<script>
let r = 153;
let g = 153;
let b = 153;

function changeBG() {
  r = Math.floor(Math.random() * 255);
  g = Math.floor(Math.random() * 255);
  b = Math.floor(Math.random() * 255);
}
</script>
```

DOM und Virtual DOM

Frameworks nutzen unterschiedliche Wege, um Websites zu aktualisieren.

■ Was ist schneller?

Das Document Object Model (DOM) ist die Schnittstelle zwischen HTML und JavaScript. Alle HTML-Elemente sind darin Objekte in einer Baumstruktur, die verändert, hinzugefügt oder gelöscht werden können. Eine Anwendung kann langsam werden, wenn mehrere Stellen im DOM einzeln geändert werden. Daher arbeiten Frameworks wie React und Vue.js mit einem Virtual DOM, einer Abbildung der jeweiligen UI-Komponente in JavaScript. Änderungen werden zunächst im Virtual DOM durchgeführt, was wesentlich schneller geht. Der neue Stand wird dann auf das reale DOM angewendet. Svelte-Erfinder Rich Harris argumentiert, dass ein Virtual DOM purer Overhead sei (**bit.ly/3g37nVT**). Andere Entwickler meinen, dass Svelte ein Virtual DOM brauche (**bit.ly/3cnSQId**).

Svelte bietet ein sehr gutes Tutorial, das Ihnen in kleinen Kapitelchen die verschiedenen Mechanismen erklärt. Sie können diese direkt im Browser testen.

Zusätzlich finden Sie auf der Website auch eine Reihe von Beispielen, die Sie mit einem Klick schnell in der REPL forken und verändern können.

In der Umfrage *State of JavaScript 2020* lag Svelte im Bereich *Satisfaction* mit 89 Prozent knapp vor React mit 88 Prozent (stateofjs.com).

Wir starten hier mit drei festen Werten für *r*, *g* und *b*. Das sorgt für einen Grauton als Default. Die Funktion *changeBG* berechnet einfach nur drei zufällige Zahlen. Nun müssen wir noch das HTML mit dem Script verknüpfen. Ändern Sie das HTML in:

```
<div style="background-color:
  rgb({r}, {g}, {b});">
  <button on:click={changeBG}>
    Change BGcolor!
  </button>
</div>
```

Der Einfachheit halber geben wir die Hintergrundfarbe direkt per Inline-Style an. Die Werte *r*, *g*, *b* sind hier in geschweiften Klammern geschrieben. Immer, wenn sich einer dieser Werte ändert, registriert Svelte diese Änderung und wendet den neuen Wert direkt an. Der Button wird über *on:click* mit der entsprechenden Funktion verküpft.

Vergleich von Frontend Frameworks

Was macht ein Framework besser als andere? Vergleiche sind schwierig, weil Vorkenntnisse und Vorlieben eine Rolle spielen. Wir können aber ein paar Zahlen vergleichen.

Im direkten Vergleich ist React weiter verbreitet und bietet mehr Tools und Komponenten als Svelte. Wir konzentrieren uns hier aber auf drei andere Metriken: *Performance*, *Größe der App* und *Anzahl der Codezeilen*. Mike Nikles hat ein paar einfache React-Beispiele in Svelte nachgebaut und dabei zwischen 35 und 80 Prozent an Codezeilen gespart (**bit.ly/3cnTOOH**). Doch gilt das auch für größere Apps?

Conduit ist ein Klon von medium.com, der mit verschiedenen Frameworks umgesetzt wurde.

RealWorld App

Die RealWord App beschreibt sich als „Mutter aller Demo-Apps“. Es ist ein Klon von **medium.com**, der mit verschiedenen Front- und Backend Frameworks umgesetzt wurde (**bit.ly/2RrSUZR**). So können Sie die Umsetzung einer realen App in Frameworks wie React, Angular, Elm, Vue.js oder auch Svelte miteinander vergleichen.

A RealWorld Comparison 2020

Entwickler Jacek Schae hat sich diese Umsetzungen in den letzten Jahren immer wieder genauer angesehen (**bit.ly/3wYNIMc**). Die *Performance* wurde mit Lighthouse getestet. Dabei erreichte Svelte eine Wert von 99 (Vue: 86, React + MobX: 82). Bei der Größe ging es um gzipped Files der App inkl. aller Abhängigkeiten. Hier kam Svelte auf 15kB (Vue: 71 kB, React + MobX: 97,2 kB). Bei der Anzahl an Codezeilen lag Svelte bei 1.057 (React + MobX: 1.917, Vue: 2.076). Weniger Code bedeutet in der Regel auch: bessere Wartbarkeit. Sie finden im Web weitere Vergleiche, bei denen Svelte ähnlich gut abschneidet.

Unsere kleine RandomBG-Komponente können Sie online im REPL testen und forken (bit.ly/3crbhp3).

Im ersten Reiter A*pp.svelte* möchten wir die neue Komponente nun verwenden. Notieren Sie hier:

```
<script>
import RandomBG from
  './RandomBG.svelte';
</script>

<RandomBG />
```

Das importiert die neue Komponente und baut sie dann direkt auf der Webseite ein. Wenn Sie nun den Button drücken, ändert sich tatsächlich die Hintergrundfarbe zufällig. Genau genommen wechselt der Farbton animiert innerhalb von 0,3 Sekunden zur neuen Farbe, weil wir das im CSS über die Transition so festgelegt haben.

Komponenten sind dazu gedacht, dass sie mehrfach verwendet werden können. Ergänzen Sie in der *App.svelte* ein zweites Element *<RandomBG />*. Ein paar Klicks demonstrieren, dass die Komponenten automatisch unabhängig voneinander arbeiten, ohne dass wir in der *RandomBG.svelte* darauf achten müssen, was wohl passiert, wenn mehrere Komponenten benutzt werden. Ebenso gelten die CSS-Einstellungen für das div in der *RandomBG.svelte* nur für das div in dieser Komponente – anders ausgedrückt: Das CSS ist *scoped*. Würden Sie in der *App.svelte* ein weiteres div einbauen, übernimmt es also nicht die Eigenschaften des *RandomBG*-divs.

Ein Blick in den Quellcode zeigt, dass Svelte dem div der Komponente eine spezielle Klasse gibt, hier: *svelte-ze1c3d*. Die Kombination *ze1c3d* basiert auf einem Hashwert für die Styles der Komponente. Solch eine Klasse erfüllt den Zweck, aber wenn Ihre App mehrere solcher Klassen enthält, wird es schwieriger, den Code über die Developer Tools zu analysieren. Sie können alternativ aber auch mit eigenen Klassen – zum Beispiel nach dem BEM-Schema (**getbem.com**) – arbeiten und die notwenigen Eigenschaften in eine globale CSS-Datei schreiben.

Defaults und Properties

Zunächst sind Variablen nur in der jeweiligen Komponente zugänglich. Aber vielleicht möchten Sie Daten von einer zur anderen übergeben. Dafür können Sie über *export* sogenannte *properties*, kurz *props*, anlegen. In unserer Komponente hat die Hintergrundfarbe Defaultwerte für *r*, *g* und *b*, die zusammen Grau ergeben. Ändern Sie die ersten Zeilen im Script in:

```
export let r = 153;
export let g = 153;
export let b = 153;
```

Nun können Sie diese Werte beim Aufruf der Komponente optional überschreiben.

```
<RandomBG />
<RandomBG r={164} g={212} b={180} />
```

Hier erhält die erste Komponente die Defaultwerte, die zweite Komponente erscheint aber mit einem grünen Hintergrund. Diesen Stand können Sie sich in der REPL unter **bit.ly/3crbhp3** ansehen.

Variante mit Bindings

Nun basteln wir eine kleine Variante der Komponente. Die Hintergrundfarbe soll sich nicht mehr zufällig ändern, stattdessen soll es drei Schieberegler geben, mit denen wir die einzelnen Werte für RGB direkt beeinflussen wollen. Sie können hier entweder den aktuellen Code forken und die bisherige Komponente überschreiben; oder aber Sie schreiben eine weitere Komponente in einem neuen Reiter *RangeBG.svelte*.

Zunächst einmal können Sie die Funktion *changeBG()* aus dem Script löschen. Den Button im Markup ersetzen Sie durch:

```
<input type=range bind:value={r}
  min=0 max=255>
<input type=range bind:value={g}
  min=0 max=255>
<input type=range bind:value={b}
  min=0 max=255>
```

Im CSS ergänzen Sie nun nur noch:

```
div { flex-direction: column;
  /* Rest wie zuvor */  }
input { width: 50%; }
```

Auf bit.ly/3wZyerk hat die Svelte Society ein nützliches Cheat Sheet angelegt.

Die Website madewithsvelte.com bietet eine Sammlung von Boilerplates, Dev Tools und UI-Komponenten, die mit Svelte erstellt wurden.

Das war es auch schon. Nun können Sie die Farbe über die drei Regler bestimmen. Das besondere ist hierbei die Verknüpfung über *bind:value*. Dieses *Binding* funktioniert in beide Richtungen: Wenn Sie die Regler verschieben, ändert sich die Hintergrundfarbe im div. Wenn Sie aber die Defaultwerte über Props ändern (so wie oben), passen sich die Positionen der Regler automatisch den neuen Werten an (REPL: **bit.ly/2TERax5**).

Sicher einen Blick wert

Unsere Beispiel-Komponenten könnten Sie natürlich auch sofort komplett in Vanilla JavaScript schreiben. Das hängt davon ab, wie gut Sie sich mit JavaScript auskennen. Frameworks wie Svelte nehmen Ihnen jedoch einiges an Arbeit ab, weil Sie sich nicht weiter mit Event Handlern, Bindings oder Scoping herumschlagen müssen. Auch wenn einige Elemente etwas seltsam erscheinen, etwa das *export* bei den props, ist der Code einsteigerfreundlicher als bei anderen Frameworks. Natürlich bietet Svelte viel mehr als in diesen Beispielen zu sehen ist, etwa *Reactive Declarations*, *Reactive Statements*, *Logik-* und *Await-Blöcke* und *Event Modifier*. In der nächsten Ausgabe nutzen wir einige dieser Features für eine etwas größere Svelte-Komponente. ■

Wenn Sie Visual Studio Code als Code Editor verwenden, bietet sich die Erweiterung *Svelte for VS Code* an.