Vue.js Miniintro

Inhalt

Was ist Vue.js?

> *Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.*

Quickstart

Der einfachste Weg zum starten ist, Vue.js vom CDN einbinden und los gehts.

```
<!-- Loading vue from a CDN -->
<script src="https://unpkg.com/vue@next"></script>
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>My first Vue app</title>
  <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
  <div id="hello-vue">
    {{ message }}
  </div>

  <script>
    const HelloVueApp = {
      data() {
        return {
          message: 'Hello Vue!!'
        }
      }
    }

    Vue.createApp(HelloVueApp).mount('#hello-vue')
```

```
  </script>
</body>
</html>
```

Demo 🤯

TODO: Demo is broken an dmost likel won't work with the corrent CodeSandbox setup.

- Hello Vue!!

Nutzung per npm

Vue.js kann auch als npm-package genutzt werden. Dafür muss `npm install vue` ausgeführt werden, danach kann vue importiert werden.
Als Template verwenden wir das gleiche wie beim vorigen Beispiel.

```
// main.js
import Vue from 'vue';

const HelloVueApp = {
  data() {
    return {
      message: 'Hello Vue!!'
    }
  }
}

Vue.createApp(HelloVueApp).mount('#hello-vue')
```

TODO Add Demo TODO: Add Codesandbox

Dekleratives Rendering

Im Herzen von Vue ist ein System, welches das Deklarative Rendering ermöglicht. Wie im ersten Beispiel ersichtlich, ist die Template Syntax einfach zu verstehen. Dabei sind unsere variablen **reactive**. Dies bedeutet, dass alle Änderungen an unserem internen state selbständig im DOM angepasst werden.

```
<!-- html -->
<div id="counter">
  Counter: {{ counter }}
</div>
```

```
// js
const Counter = {
  data() {
    return {
      counter: 0
    }
  },
  mounted() {
    setInterval(() => {
```

```
      this.counter++
    }, 1000)
  }
}

Vue.createApp(Counter).mount('#counter')
```

Demo 🤯

- TODO: Demo is broken
- Dekleratives Rendering & Lifecycle hooks

In dem Beispiel wird das DOM jede Sekunde mit dem neuen counter updated.
Hier seht ihr eine der Lifecycle hooks `mounted()`. Diese sind ähnlich wie z.B. `componendDidMount()` in React und können genutzt werden um einen Code auszuführen, wenn eine Instanz dieser Vue-Komponente den angegebenen Lebenszyklus durchläuft.

Zudem können wir auch unseren State auf den Attribut eines Elementes binden. Dies erreichen wir mit der `v-bind` Directive. Dies gibt es auch als Kurzversion indem wir einfach nur `:` verwenden. Zu beachten ist, sobald wir eine Directive nutzen (generell immer `v-something`), schreiben wir JavaScript innerhalb der `""`.

```
<!-- html -->
<div id="bind-attribute">
  <p v-bind:title="message">
    Hover your mouse over me for a few seconds to see my dynamically bound
    title!
  </p>
  <br>Same same
  <p :title="message">
    Hover your mouse over me for a few seconds to see my dynamically bound
    title!
  </p>
  <br>Stringinterpolation
  <p :title="'Hello,' + message">
    Hover your mouse over me for a few seconds to see my dynamically bound
    title!
  </p>
</div>
```

```
// js
const AttributeBinding = {
  data() {
    return {
      message: 'You loaded this page on ' + new Date().toLocaleString()
    }
  }
}

Vue.createApp(AttributeBinding).mount('#bind-attribute')
```

Demo 🤯

- Binding Attributes

Hilfreiche Links

- Reactivity in Depth 🔥
- Declerative Rendering

User Input

Die Events, die von unserem UI abgeschossen werden, können mit der `v-on` Directive abgefangen werden. Auch hier gibt es eine gekürzte Version mit `@`.

Im folgenden Beispiel ist zu sehen wie wir einen Counter über einen Button hochzählen. Dabei ist auch das neue `methods`-Objekt zu sehen. Dieses beinhaltet unsere **Methoden**, welche wir von anderen Methoden, unseren Lifecycle hooks oder auch von unserem Template aus ausrufen können.

```
<!-- html -->
<div id="event-handling">
  {{ counter }}
  <button v-on:click="increment">Increment</button>
  Same same
  <button @click="increment">Increment</button>
</div>
```

```
// js
const EventHandling = {
  data() {
    return {
      counter: 0
    }
  },
  methods: {
    increment() {
      this.counter++;
    }
  }
}

Vue.createApp(EventHandling).mount('#event-handling')
```

Demo 🤯

- Eventhandling

Two-Way Databinding

Mit der `v-model` Directive können wir unseren State von den Komponenten direkt durch das UI manipulieren.

```html
<!-- html -->
<div id="two-way-binding">
  <p>{{ message }}</p>
  <input v-model="message" />
</div>
```

```js
// js
const TwoWayBinding = {
  data() {
    return {
      message: 'Hello Vue!'
    }
  }
}

Vue.createApp(TwoWayBinding).mount('#two-way-binding')
```

Demo 🤯

- Two-Way Databinding

Conditionals and Loops

Konditionales Rendering

Konditionales Rendering ist sehr einfach zu implementieren mit der `v-if` Directive. Diese kann ebenfalls mit `v-else-if` und/oder `v-else` kombiniert werden. Bei der Verwendung dieser Directives wird das ELement aus dem DOM entfernt, solange es nicht angezeigt wird.
Mit der `v-show` Directive kann ein Element mit `display: none;` lediglich ausgeblendet werden. Somit bleibt es im DOM vorhanden.

Die Elemente die dabei angezeigt bzw. ausgeblendet werden, können dazu animiert werden. Siehe dafür die Dokumentation für die Transitions.

```html
<!-- html -->
<div id="conditional-rendering">
  <span v-if="showText">Hello there</span>
  <button @click="showText = !showText">Toggle Text</button>
</div>
```

```js
// js
const ConditionalRendering = {
  data() {
    return {
      showText: true
    }
  }
}
```

```
Vue.createApp(ConditionalRendering).mount('#conditional-rendering')
```

Demo 🤯

- Konditionales Rendering

Loops

Mit der `v-for` Directive können wir durch Arrays oder auch Objekte loopen. Dabei wir die Direktive auf dem Element angebracht, welches wiederholt werden soll.
Mehr zum Rendering von Listen findest Du hier.

```
<!-- html -->
<div id="list-rendering">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

```
// js
const ListRendering = {
  data() {
    return {
      todos: [
        { text: 'Learn JavaScript' },
        { text: 'Learn Vue' },
        { text: 'Build something awesome' }
      ]
    }
  }
}

Vue.createApp(ListRendering).mount('#list-rendering')
```

Demo 🤯

- Loops

Components

Vue.js ist gleich wie andere MV*-Frameworks aufgebaut, so dass man sein Widget/Webapp oder ähnliches in verschiedenen Komponenten aufsplitten kann. Diese können ebenfalls ineinander verschachtelt werden. Das Registrieren von Komponenten ist einfach. Wir erstellen unsere Komponenten und registrieren diese in unserer App-Instanz oder in anderen Komponenten. Das Registrieren wird über den `components` Key gemacht.

```
<!-- html -->
<div id="app">
```

```
  <ul>
    <!-- Create an instance of the todo-item component -->
    <todo-item></todo-item>
  </ul>
</div>
```

```
// js
const IconHot = {
  template: `🔥`
}

const TodoItem = {
  template: `<li>This is a todo <icon-hot></icon-hot></li>`,
  components: {
    IconHot, // Register a new component
  },
}

// Create Vue application
const app = Vue.createApp({
  components: {
    TodoItem, // Register a new component
  },
  // ... Other properties for the component
})

// Mount Vue application
app.mount("#app")
```

Demo 🤯

- Components

Practice 🔥

Öffne diese **CodeSandbox** als Startpunkt.

Füge die fehlende Funktionalität hinzu, damit unsere Todo-App deployt werden kann.

Zeit: ~ 15 min

Solution: https://codesandbox.io/s/g5br9

Vorteile von Vue.js

Einer der grössten Vorteile von Vue.js gegenüber React ist, dass Vue.js auch ein bestehendes Template in eine Vue-Instanz umwandeln kann. Somit kann z.B. alles relevante HTML, welches für SEO genützt wurde, bereits von einem Server gerendert werden, ohne dass wir Vue.js auf dem Server ausführen müssen. Dies hatten wir bei der Navigation von css.ch zu unserem Vorteil genutzt. Somit konnten wir alle SEO-Anforderungen erfüllen, konnten jedoch alle JS-Funktionalität, welche die Navigation benötigt, mit Vue abwickeln. Wenn man den Source-Code von css.ch ansieht, ist dies gut ersichtlich.

Weiterführende Themen

- Getting started Guide
- Composing with Components
- Single File Components 🔥
- Composition API 🔥
- Vite - Next Generation Frontend Tooling 🔥