LU09b – Vue.js: Reaktive Variablen, v-bind und Mustache

Lernziele

- Sie verstehen, was reaktive Variablen sind, wie `ref()` funktioniert und warum wir sie brauchen.
- Sie kennen die wichtigsten Vue-Direktiven: `v-bind` und `v-on`.
- Sie können JavaScript-Werte direkt im HTML-Template mit `{{ }}` ausgeben.
- Sie sind vertraut mit der Struktur einer Vue-Komponente (`<script setup>`, `<template>`, `<style>`).

Projektaufbau: Was ist was?

Nach dem Setup mit Vue CLI entsteht folgende Ordnerstruktur:

```
my-first-vue-app/
├── src/
│   ├── App.vue          ← Haupt-Komponente, rendert alles
│   ├── components/      ← Eigene Komponenten kommen hier
rein
│   └── assets/          ← Bilder, Fonts, globale CSS-
Dateien
├── index.html           ← Einstiegspunkt der App
└── package.json         ← Projektinformationen und
Dependencies
```

`App.vue` ist das Fenster zu unserer Applikation. Alles, was auf der Seite erscheint, wird direkt oder indirekt von `App.vue` gerendert.

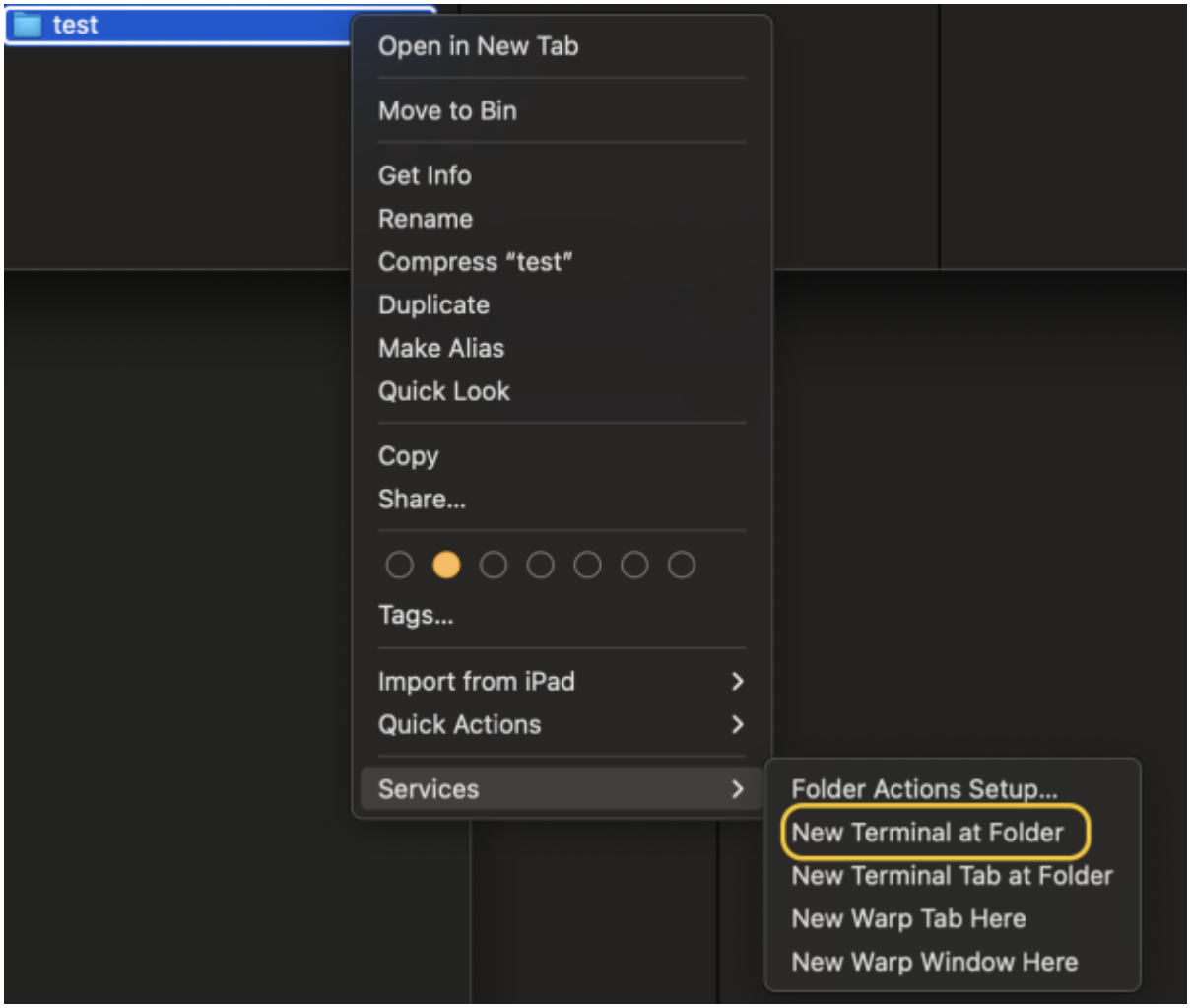
Vue-Projekt erstellen

1. Projekt anlegen

Öffnen Sie das Terminal im Ordner, in dem Sie das Projekt erstellen möchten:

- **Mac:** Rechtsklick auf den Ordner im Finder → *Dienste* → *Neues Terminal beim Ordner*
- **Windows 11:** Rechtsklick auf den Ordner im Explorer → *In Terminal öffnen*

- **Windows 10:** Rechtsklick auf den Ordner im Explorer → *PowerShell-Fenster hier öffnen* (falls nicht sichtbar: Shift + Rechtsklick)




```
npm create vue@latest
```

Folgen Sie dem Setup-Assistenten:

- **Project name:** `my-first-vue-app`
- **TypeScript:** No
- **Features:** Nichts auswählen (Enter)
- **Experimental features:** Nichts auswählen (Enter)
- **Skip all example code:** Yes

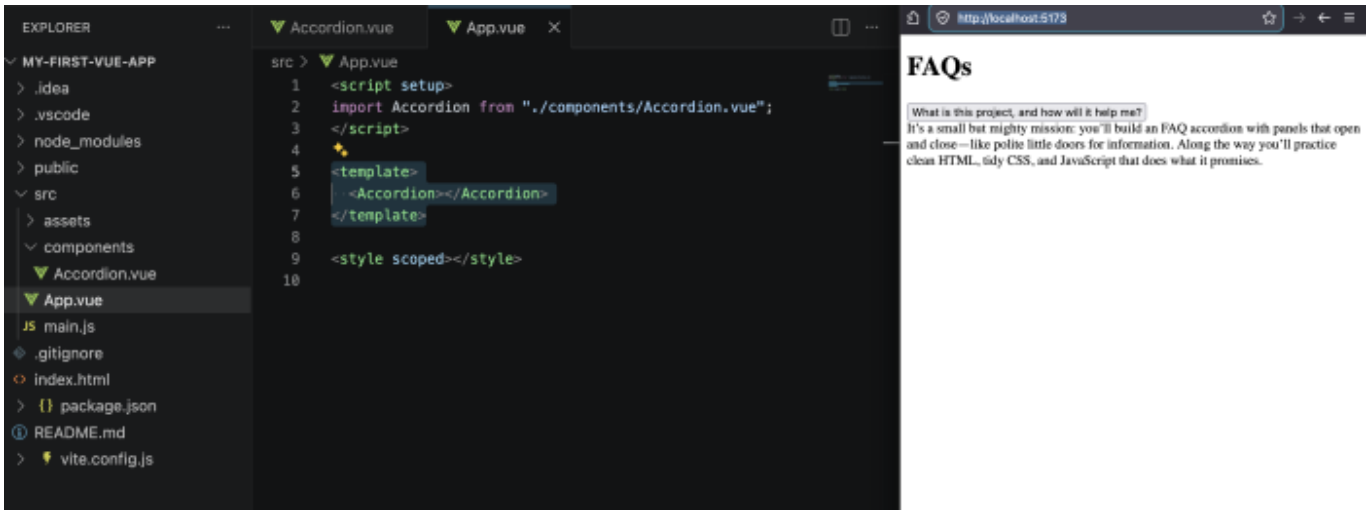
```
291_WebUI/09_LU/test using ☁ temp-config/
➜ npm create vue@latest

> npx
> create-vue

┌  Vue.js - The Progressive JavaScript Framework
│
◇  Project name (target directory):
│  my-first-vue-app
│
◇  Use TypeScript?
│  No
│
◇  Select features to include in your project: (↑/↓ to navigate, space to select, a to
│  toggle all, enter to confirm)
│  none
│
◇  Select experimental features to include in your project: (↑/↓ to navigate, space to
│  select, a to toggle all, enter to confirm)
│  none
│
◆  Skip all example code and start with a blank Vue project?
│  ● Yes / ○ No
└
```

2. Projekt starten

```
cd my-first-vue-app
npm install
npm run dev
```

```
291_WebUI/09_LU/test using ☁ temp-config/ took 1m 14.9s
➜ cd my-first-vue-app

09_LU/test/my-first-vue-app via ● v20.19.6 using ☁ temp-config/
➜ npm install
npm warn ERESOLVE overriding peer dependency
npm warn ERESOLVE overriding peer dependency
npm warn ERESOLVE overriding peer dependency

added 117 packages, and audited 118 packages in 7s

30 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

09_LU/test/my-first-vue-app via ● v20.19.6 using ☁ temp-config/ took 6.8s
➜ npm run dev
```

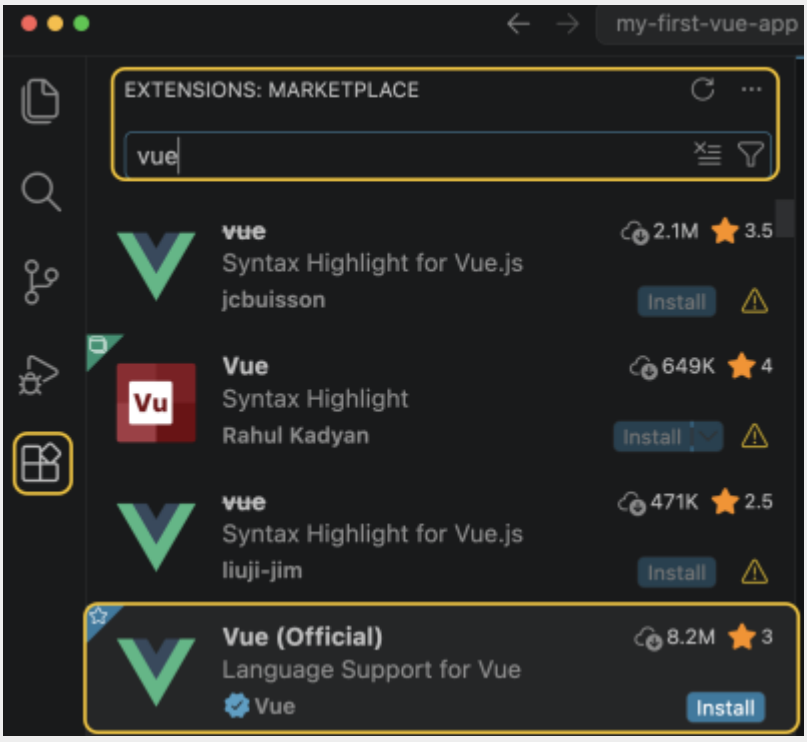
Öffnen Sie http://localhost:5173/ im Browser.

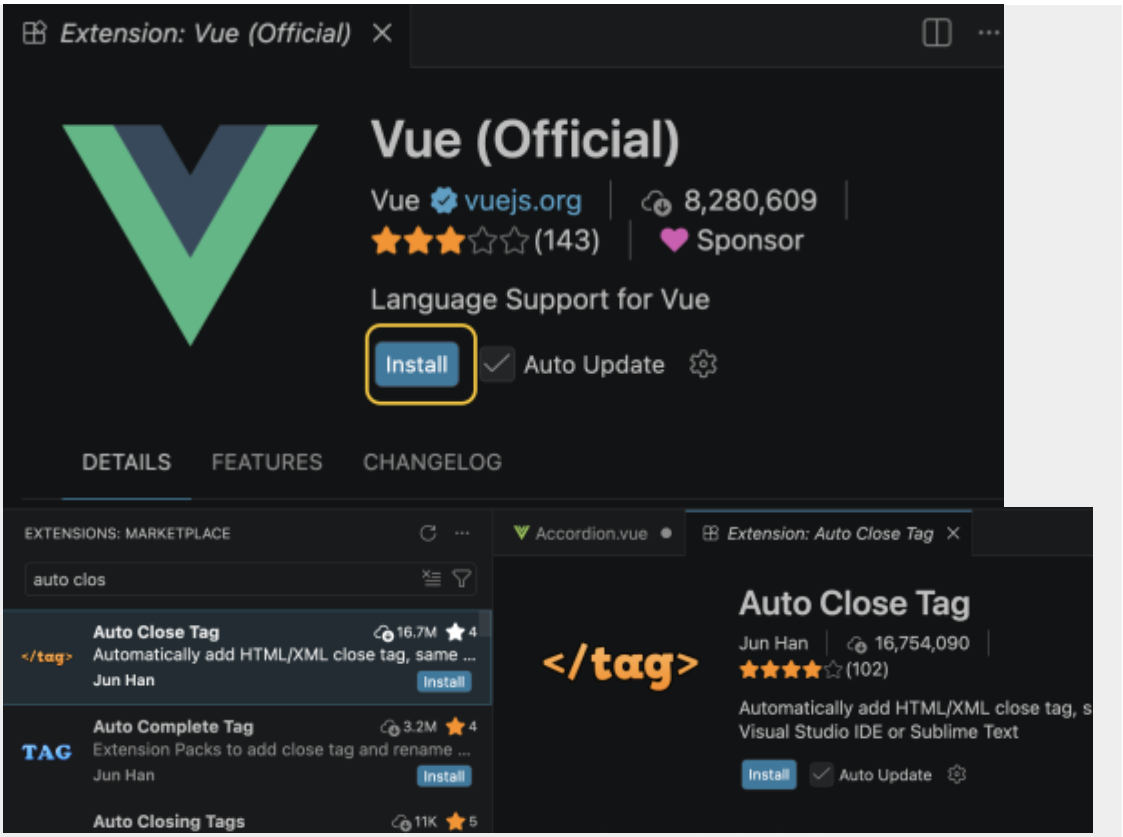


3. VS Code einrichten

Damit Syntax-Highlighting funktioniert, installieren Sie zwei Extensions in VS Code (Webstorm braucht das nicht):

- **Vue (Official)** – Syntax-Highlighting für `.vue`-Dateien
- **Auto Close Tag** – schliesst HTML-Tags automatisch





FAQ Accordion in Vue

Schritt 1: Komponente erstellen

Erstellen Sie im `src/`-Ordner zwei neue Unterordner: `components/` und `assets/`.

Erstellen Sie darin die Datei `components/Accordion.vue` mit folgendem Grundgerüst:

```
<script setup>

</script>

<template>

</template>

<style>

</style>
```

Accordion.vue erstellen

Fügen Sie im `<template>`-Bereich das HTML-Grundgerüst ein – Sie können es aus dem bestehenden FAQ-Accordion übernehmen:

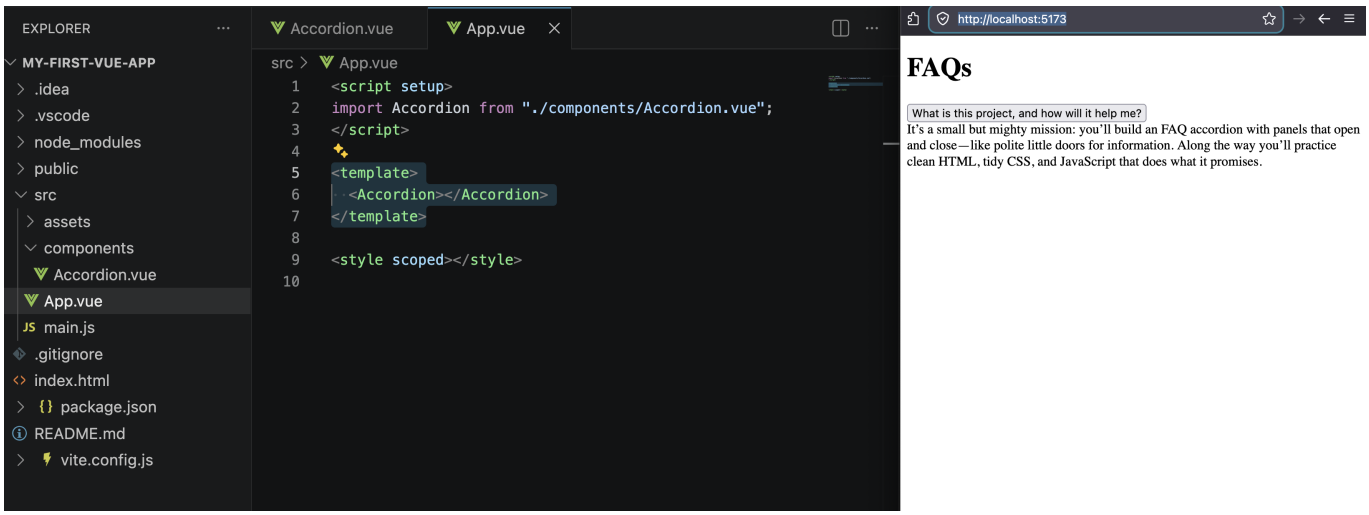
```
<template>
  <div class="accordion">
    <h1>FAQs</h1>
    <button class="accordion-btn">
      What is this project, and how will it help me?
    </button>
    <div class="panel">
      It's a small but mighty mission: you'll build an FAQ
accordion with
      panels that open and close.
    </div>
  </div>
</template>
```

Schritt 2: Komponente in App.vue einbinden

Damit die Komponente im Browser erscheint, muss sie in `App.vue` importiert und gerendert werden:

```
<script setup>
import Accordion from './components/Accordion.vue';
</script>

<template>
  <Accordion />
</template>
```

Was haben wir gemacht? Wir haben eine eigene **Komponente** erstellt – ein abgeschlossenes UI-Baustein, der einmal gebaut und beliebig oft eingesetzt werden kann. `App.vue` importiert und rendert ihn. Das ist das Komponentenprinzip von Vue.



Schritt 3: CSS hinzufügen

Im `<style>`-Bereich von `Accordion.vue` fügen wir die bekannten CSS-Regeln ein:

```
.panel {
  display: none;
}

.panel.open {
  display: block;
}
```

Schritt 4: Interaktivität – zuerst mit einer normalen Variable

Wir versuchen es zuerst mit einer gewöhnlichen Variable:

```
<script setup>
let isOpen = false;

function toggleOpen() {
  isOpen = !isOpen;
  console.log('isOpen', isOpen);
}
</script>

<template>
  <div class="accordion">
    <h1>FAQs</h1>
    <button v-on:click="toggleOpen" class="accordion-btn">
      What is this project, and how will it help me?
    </button>
    <div class="panel" v-bind:class="{ open: isOpen }">
      It's a small but mighty mission...
    </div>
  </div>
</template>
```

Testen Sie im Browser: Die Console zeigt den richtigen Wert – aber die Klasse `open` wird nicht gesetzt. **Warum?**

Die Variable ist nicht reaktiv. Vue weiss nicht, dass sie sich geändert hat, und rendert das Template nicht neu.

Schritt 5: ref() – die reaktive Variable

```
import { ref } from 'vue';
const isOpen = ref(false);
```

`ref()` macht die Variable reaktiv – Vue beobachtet sie. Ändert sich ihr Wert, aktualisiert Vue automatisch alle Stellen im Template, die davon abhängen.

Wichtig: Innerhalb von JavaScript greift man auf den Wert mit `.value` zu:

```
function toggleOpen() {
  isOpen.value = !isOpen.value;
}
```

Im `<template>` verwendet man `isOpen` direkt – ohne `.value`.

Schritt 6: Mustache {{ }} – JavaScript direkt im HTML

Mit doppelten geschweiften Klammern können JavaScript-Werte direkt ins HTML eingebettet werden:

```
<script setup>
const faqTitle = 'FAQs';
</script>

<template>
  <h1>{{ faqTitle }}</h1>
</template>
```

`{{ faqTitle }}` wird durch den aktuellen Wert der Variable ersetzt. Ändert sich der Wert (bei einer `ref()`-Variable), aktualisiert sich die Anzeige automatisch.

Finaler Code

```
<script setup>
import { ref } from 'vue';

const isOpen = ref(false);
const faqTitle = 'FAQs';

function toggleOpen() {
  isOpen.value = !isOpen.value;
}
</script>

<template>
  <div class="accordion">
    <h1>{{ faqTitle }}</h1>
    <button v-on:click="toggleOpen" class="accordion-btn">
      What is this project, and how will it help me?
    </button>
    <div class="panel" v-bind:class="{ open: isOpen }">
      It's a small but mighty mission: you'll build an FAQ
accordion with
```

```
      panels that open and close.
    </div>
  </div>
</template>

<style>
.panel {
  display: none;
}
.panel.open {
  display: block;
}
</style>
```

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/de/modul/m291/learningunits/lu09/theorie/b_live_coding

Last update: **2026/04/12 14:25**