LU10 – Komponenten, v-for und Props

Lernziele

- Sie können ein Array of Objects mit `ref()` erstellen und im Template ausgeben.
- Sie können mit `v-for` dynamisch über ein Array iterieren und Komponenten rendern.
- Sie können Daten via Props von einer Eltern- an eine Kind-Komponente übergeben.

Repetition: Was wir in LU09 gelernt haben

Video: ref(), Mustache-Syntax und Komponentenstruktur

Video: ref, Mustache und script/template
Aus dem Vue Mastery Crash Course (Englisch, deutsche Untertitel, 2:35 Min.)
Das Video zeigt, wie eine Vue-Komponente aufgebaut ist, was `ref()` macht und wie die Mustache-Syntax `{{ }}` Daten im Template anzeigt.

Eine kurze Übersicht des Gelernten:

Konzept	Was es macht	Beispiel
`ref()`	Erstellt eine reaktive Variable	`const isOpen = ref(false)`
`{{ }}`	Zeigt einen JS-Wert im Template an	`{{ faqTitle }}`
`:` (v-bind)	Bindet ein HTML-Attribut an einen JS-Wert	`:class=„{ open: isOpen }“`
`@` (v-on)	Reagiert auf ein Event	`@click=„toggleOpen“`

Von einem Objekt zu einem Array of Objects

In LU09 haben wir die Frage und Antwort in einem einzelnen Objekt gespeichert:

```
// LU09 – ein einzelnes Objekt
const faqData = {
  question: 'What is this project?',
  answer: "It's a small but mighty mission..."
}
```

Für ein echtes Accordion mit mehreren Fragen brauchen wir ein **Array von Objekten**:

```
import { ref } from 'vue';

const faqItems = ref([
  {
```

```
    id: 1,
    question: 'What is this project, and how will it help
me?',
    answer: "It's a small but mighty mission..."
  },
  {
    id: 2,
    question: 'Is this free?',
    answer: 'Yes. No coins, no secret handshake...'
  },
  {
    id: 3,
    question: 'Can I use this project in my portfolio?',
    answer: 'Absolutely. Show it off proudly...'
  },
])
```

Jedes Objekt hat drei Eigenschaften: `id`, `question` und `answer`. Die `id` brauchen wir später für das `key`-Attribut bei `v-for`.

v-for: Dynamisch über ein Array iterieren

Video: v-for und Arrays

Video: v-for und Arrays
Aus dem Vue Mastery Crash Course (Englisch, deutsche Untertitel, 2:52 Min.)
Kontext: Im Video wird ein fiktiver Online-Shop für Socken als Beispiel verwendet. Das gezeigte Konzept – `v-for` über ein Array – funktioniert aber identisch in unserem FAQ Accordion. Die Parallele:

Socks-App	**Unser Accordion**
`detail in details`	`item in faqItems`
`{{ detail }}`	`{{ item.question }}`

`v-for` funktioniert wie eine `forEach`-Schleife in JavaScript – nur direkt im HTML:

```
// JavaScript forEach
faqItems.forEach(item => {
  console.log(item.question);
});
```

```
<!-- Vue v-for – dasselbe Konzept im Template -->
<div v-for="item in faqItems" :key="item.id">
  {{ item.question }}
</div>
```

Vue rendert für jedes Element im Array automatisch ein neues `<div>`. Fügen wir dem Array ein neues Objekt hinzu, erscheint sofort ein neues Element im Browser.

Warum braucht auch :faq einen Doppelpunkt?

```
<!-- Ohne Doppelpunkt: übergibt den Text "item" als String -
->
<AccordionItem faq="item" />

<!-- Mit Doppelpunkt: übergibt die JavaScript-Variable
"item" -->
<AccordionItem :faq="item" />
```

Ohne `:` erhält `AccordionItem` buchstäblich den Text **„`item`“** – vier Zeichen, weiter nichts. Mit `:` wertet Vue den Ausdruck als JavaScript aus und übergibt das tatsächliche Objekt aus der `v-for`-Schleife.

Das key-Attribut

Bei `v-for` sollte jedes Element einen eindeutigen Schlüssel erhalten. Das hilft Vue, die Elemente effizient zu verwalten:

```
<div v-for="item in faqItems" :key="item.id">
  {{ item.question }}
</div>
```

`:key` braucht einen eindeutigen Wert – idealerweise eine ID aus den Daten oder der Datenbank.

Das Problem mit einem gemeinsamen State

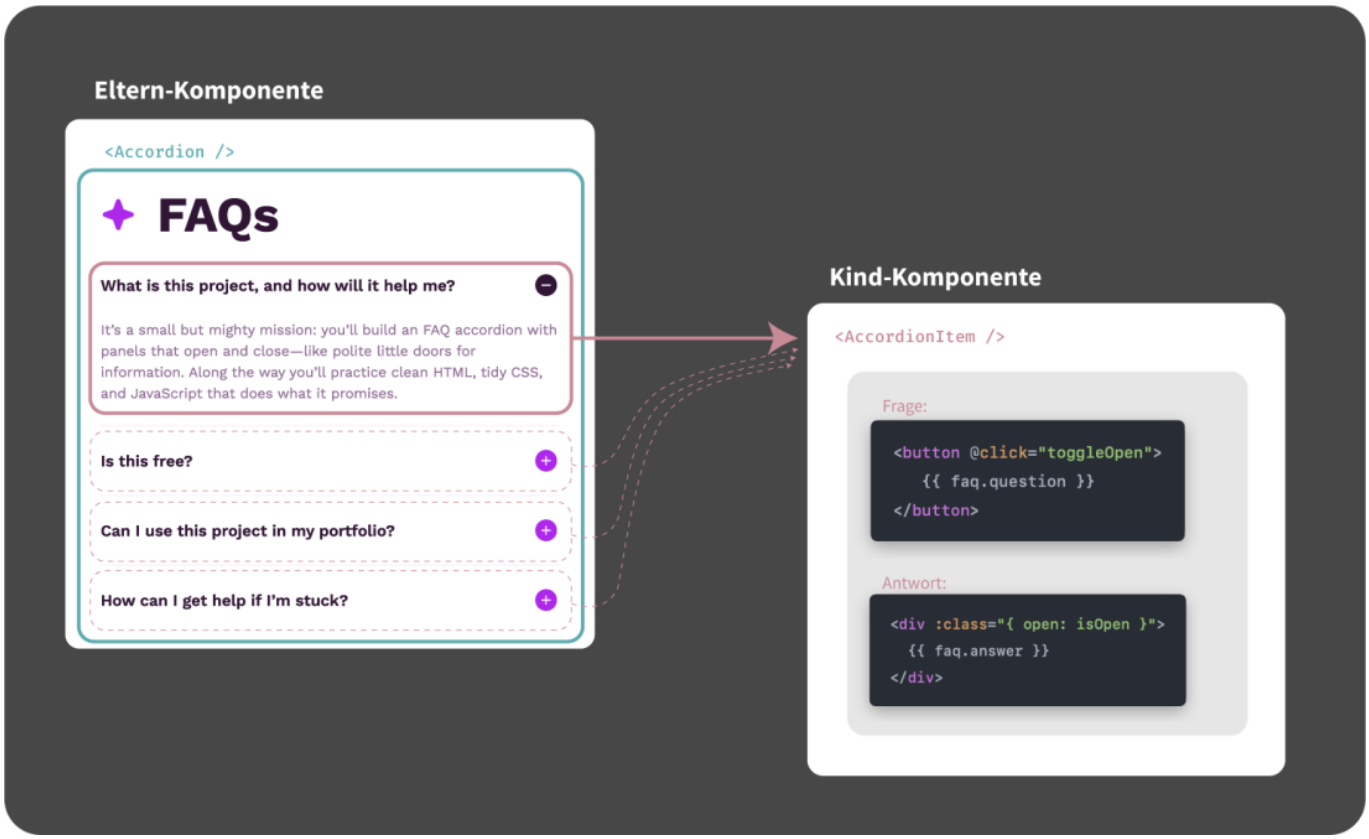
Wenn wir `v-for` direkt im Accordion verwenden und `isOpen` im Parent gesetzt wird, teilen alle Items denselben Zustand – klickt man auf eine Frage, reagieren alle gleichzeitig:

```
<!--  Problem: isOpen ist für alle Items gleich -->
<div v-for="item in faqItems" :key="item.id">
  <button @click="toggleOpen">{{ item.question }}</button>
  <div class="panel" :class="{ open: isOpen }">{{
item.answer }}</div>
</div>
```

Die Lösung: Jede Frage-Antwort-Einheit bekommt ihren **eigenen State** – durch eine eigene Komponente.

Komponenten: Wiederverwendbare Bausteine

Eine **Komponente** ist ein abgeschlossener UI-Baustein mit eigenem Template, eigenem Script und eigenem Style. Sie kann beliebig oft verwendet werden.



Jede `AccordionItem`-Komponente verwaltet ihr `isOpen` selbst.

Props: Daten vom Parent zum Kind



Mit **Props** können wir Daten von einer Eltern-Komponente an eine Kind-Komponente übergeben. Man kann sich Props wie Parameter einer Funktion vorstellen:

```
<template>
//Elternkomponente mit :faq="item" <- das übergibt "item"
als Prop
<AccordionItem v-for="item in faqData" :faq="item" />
</template>

// Kind-Komponente  mit Prop
// AccordionItem.vue empfängt: faq: Object (Datentyp)
<script setup>
const props = defineProps({
  faq: Object,
});
</script>
```

Props empfangen: defineProps()

In der Kind-Komponente deklarieren wir, welche Props wir erwarten:

```
// AccordionItem.vue – <script setup>
const props = defineProps({
  faq: Object   // wir erwarten ein Objekt namens "faq"
});
```

Props übergeben: v-bind im Parent

Im Parent übergeben wir die Daten mit `:propname`:

```
<!-- Accordion.vue – <template> -->
<AccordionItem
  v-for="item in faqItems"
  :key="item.id"
  :faq="item"
/>
```

`:faq=„item“` bedeutet: Übergib das aktuelle `item` als Prop `faq` an `AccordionItem`.

Props im Template verwenden

In der Kind-Komponente greifen wir auf die Prop direkt mit ihrem Namen zu:

```
<!-- AccordionItem.vue – <template> -->
<button>{{ faq.question }}</button>
<div class="panel">{{ faq.answer }}</div>
```

Props sind read-only. Die Kind-Komponente darf eine Prop nicht direkt verändern. Daten fliessen immer von oben nach unten: Parent → Kind.

Finaler Code

AccordionItem.vue

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

const props = defineProps({
  faq: Object
});

const isOpen = ref(false);

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

```
<template>
  <div class="accordion-item">
    <button
      class="accordion-btn"
      @click="toggleOpen"
      :aria-expanded="isOpen"
    >
      {{ faq.question }}
    </button>
    <div class="panel" :class="{ open: isOpen }">
      {{ faq.answer }}
    </div>
  </div>
</template>

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

Accordion.vue

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

const faqItems = ref([
  {
    id: 1,
    question: 'What is this project, and how will it help
me?',
    answer: "It's a small but mighty mission: you'll build
an FAQ accordion..."
  },
  {
    id: 2,
    question: 'Is this free?',
    answer: 'Yes. No coins, no secret handshake...'
  },
  {
    id: 3,
    question: 'Can I use this project in my portfolio?',
    answer: 'Absolutely. Show it off proudly...'
  },
  {
    id: 4,
```

```
    question: 'How can I get help if I\'m stuck?',
    answer: 'Use the classic survival kit: check the
console...'
  },
])
</script>

<template>
  <div class="accordion">
    <h1>FAQs</h1>
    <AccordionItem
      v-for="item in faqItems"
      :key="item.id"
      :faq="item"
    />
  </div>
</template>
```

Zusätzliche Videos zur Vertiefung

Video	Inhalt	Dauer
Video 2 – v-bind	Attribute dynamisch binden	3:22 Min.
Video 3 – v-on	Events und EventListener	2:42 Min.
Video 4 – Style & Class Binding	Klassen und Styles dynamisch setzen	4:50 Min.
Video 1 – create-vue (Scrimba)	Projekt anlegen mit create-vue	3:07 Min.

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

Permanent link:
https://wiki.bzz.ch/de/modul/m291/learningunits/lu10/theorie/a_components?rev=1777844852

Last update: **2026/05/03 23:47**