

Scoped Slots

Slots sind eine fantastische Möglichkeit, Inhalte aus einer übergeordneten Komponente in eine untergeordnete Komponente einzufügen.

Wir können Slots mit Namen versehen und so den Inhalt gezielt an einen bestimmten Slot übergeben. Wird kein Name angegeben, so geht der Inhalt an den Default Slot. Für jeden Slot kannst du Inhalte angeben für den Fall, dass der Slot in der Parent Komponente nicht angesprochen wird (siehe Beispiel unten bei Slot **footer**)

Beispiel:

Child Komponente:

```
<template>
  <div>
    <div><slot name="header"></slot></div>
    <slot></slot>
    <div><slot name="footer">By Robert Baumgartner</slot></div>
  </div>
</template>
```

Parent Komponente:

```
<AppLayout>
  <template v-slot:header>
    I was made with Vue slots!
  </template>
  <AppButton> Home</AppButton>
</AppLayout>
```

Übrigens gibt es seit kurzem auch die Möglichkeit statt **v-slot** einfach **#** zu schreiben. Das macht das Ganze lesbarer.

```
<AppLayout>
  <template #header>
    I was made with Vue slots!
  </template>
  <AppButton> Home</AppButton>
</AppLayout>
```

Aufgabe 1: Was ist der Unterschied zwischen Props und Slots? Beide geben doch Daten an eine Child Komponente weiter?

Mit Scoped Slots gibt es die zusätzliche Möglichkeit Daten der Child Komponente der Parent Komponente zur Verfügung zu stellen. Aber weshalb sollte man das wollen?

Ein Beispiel (Siehe **Slots 2 Playground.zip**): Nehmen wir an, wir haben eine Komponente **ShowFriends**. Ihre Aufgabe ist es, eine Liste von Freunden anzuzeigen.

Die Komponente besitzt einen named Slot: **title**. Dort können wir den Text der Überschrift bekannt geben. Die Daten werden über ein Prop **friends** übergeben.

Robert Baumgartner

Child Komponente:

```
<template>
  <div>
    <h3>
      <slot name="title"> </slot>
    </h3>
    <div v-for="(friend, i) of friends" :key="i">
      <span>{{ friend.name }}</span>
    </div>
  </div>
</template>

<script>
export default {
  props: {
    friends: {
      type: Array,
    },
  },
};
</script>
```

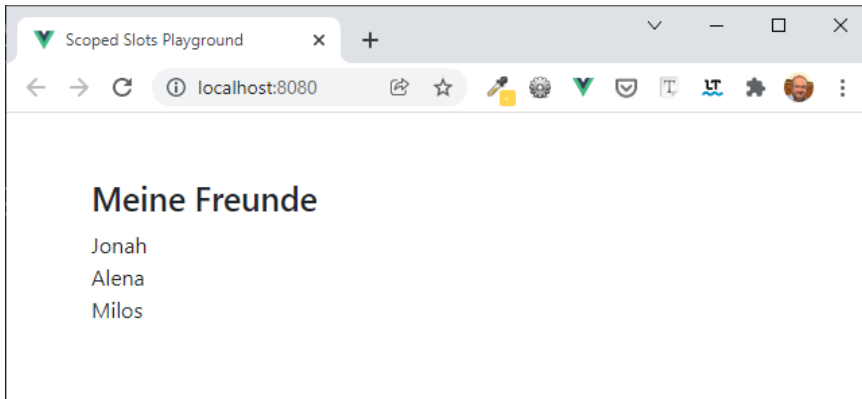
Parent Komponente:

```
<template>
  <div class="container mt-5" id="app">
    <ShowFriends :friends="myFriends">
      <template #title>Meine Freunde</template>
    </ShowFriends>
  </div>
</template>

<script>
import ShowFriends from '@components/ShowFriends.vue';

export default {
  name: 'App',
  components: {
    ShowFriends,
  },
  data() {
    return {
      myFriends: [
        { name: 'Jonah', alter: 37 },
        { name: 'Alena', alter: 28 },
        { name: 'Milos', alter: 45 },
      ],
    };
  },
};
</script>
```

Robert Baumgartner

Ausgabe:

Was aber, wenn wir dem User die Möglichkeit der Gestaltung der Zeile überlassen wollen? Was, wenn auch das Alter ausgegeben werden soll?

Wir führen in der Child Komponente einen Default Slot ein:

```
<template>
  <div>
    <h3>
      <slot name="title"> </slot>
    </h3>
    <div v-for="(friend, i) of friends" :key="i">
      <slot>
        <span>{{ friend.name }}</span>
      </slot>
    </div>
  </div>
</template>
```

In der Parent Komponente stoßen wir auf ein Problem. Wie können wir auf die Daten der aktuellen Zeile zugreifen? Die Variable **friend** gibt es nur in der Child Komponente!

```
<template>
  <div class="container mt-5" id="app">
    <ShowFriends :friends="myFriends">
      <template #title>Meine Freunde</template>
      <p>{{friend.name}} {{friend.alter}}</p>
    </ShowFriends>
  </div>
</template>
```

Wir müssen also Daten an die Parent Komponente zurückgeben. In unserem Fall die Daten der aktuellen Zeile.

Dazu binden wir die Daten an ein Slot Property. Der Name des Properties ist hier **item**, kann aber auch anders heißen.

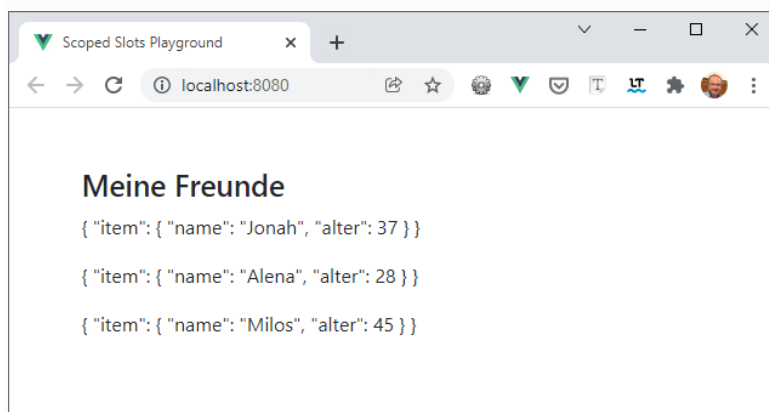
Robert Baumgartner

Child Komponente:

```
<template>
  <div>
    <h3>
      <slot name="title"> </slot>
    </h3>
    <div v-for="(friend, i) of friends" :key="i">
      <slot :item="friend">
        <span>{{ friend.name }}</span>
      </slot>
    </div>
  </div>
</template>
```

Parent Komponente:

```
<template>
  <div class="container mt-5" id="app">
    <ShowFriends :friends="myFriends">
      <template #title>Meine Freunde</template>
      <template v-slot="item">
        <p>{{item}}</p>
      </template>
    </ShowFriends>
  </div>
</template>
```

Ausgabe:

Wie du siehst, ist unser Property `item` in einem Objekt eingebunden. Daher besser gleich destruktuieren.

```
<template #row="{item}">
```

Oft verwendet man auch einen named Slot:

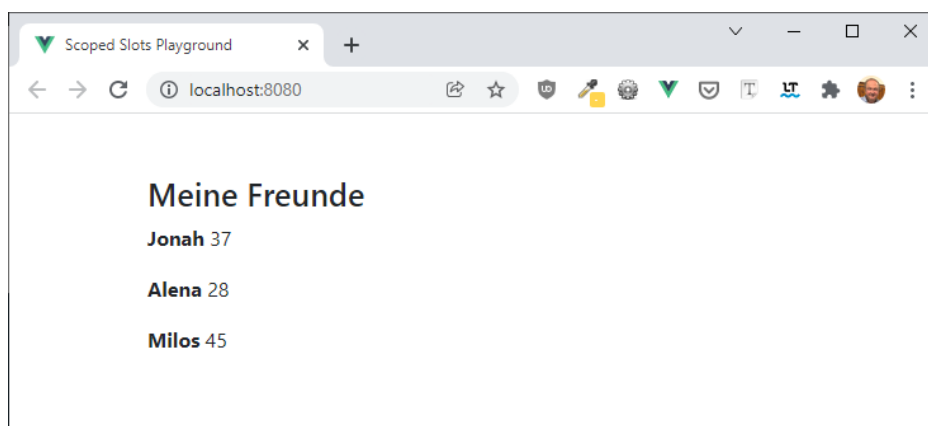
Robert Baumgartner

Child Komponente:

```
<template>
  <div>
    <h3>
      <slot name="title"> </slot>
    </h3>
    <div v-for="(friend, i) of friends" :key="i">
      <slot name="row" :item="friend">
        <span>{{ friend.name }}</span>
      </slot>
    </div>
  </div>
</template>
```

Parent Komponente:

```
<template>
  <div class="container mt-5" id="app">
    <ShowFriends :friends="myFriends">
      <template #title>Meine Freunde</template>
      <template #row="{item}"> <!-- / aka v-slot:row= {item} -->
        <p><span class="fw-bold">{{item.name }}</span> {{item.alter }}</p>
      </template>
    </ShowFriends>
  </div>
</template>
```

Ausgabe:

Was jedoch, wenn wir eine Table in der Child Komponente verwenden wollen? Wir könnten die einzelnen Zellen pro Zeile mit scoped Slots versehen!

Um die Namen der Zellen zu unterscheiden, verwenden wir `row` und die Namen der entsprechenden Properties:

`name="row.name"` und `name="row.name"`

Falls ESLint einen Fehler meldet, kann ESLint für eine Zeile ausgeschaltet werden:

```
// eslint-disable-next-line
```

Robert Baumgartner

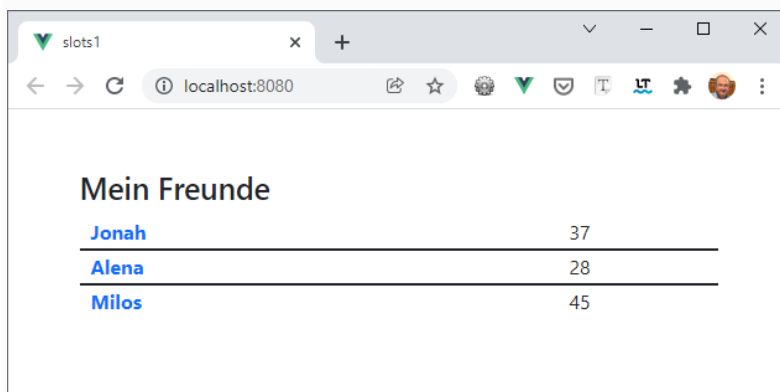
Child Komponente:

```
<template>
  <div>
    <h3>
      <slot name="title"> </slot>
    </h3>
    <table class="table">
      <tr v-for="(friend, i) of friends" :key="i">
        <td>
          <slot name="row.name" :item="friend"> </slot>
        </td>
        <td>
          <slot name="row.name" :item="friend"> </slot>
        </td>
      </tr>
    </table>
  </div>
</template>
```

Damit es übersichtlicher ist, werden die Zellen als Slot Properties definiert. Dabei wird der Name mit **.prop** eindeutig gemacht: `name="row.name"` und `name="row.alter"`.

Parent Komponente (mit alter Notation):

```
<template>
  <div class="container mt-5" id="app">
    <ShowFriends :friends="myFriends">
      <template v-slot:title>Meine Freunde</template>
      <template v-slot:row.name="{ item }">
        <span class="fw-bold text-primary"> {{ item.name }}</span>
      </template>
      <template v-slot:row.alter="{ item }">
        {{ item.alter }}
      </template>
    </ShowFriends>
  </div>
</template>
```

Ausgabe:

Jonah	37
Alena	28
Milos	45

Robert Baumgartner

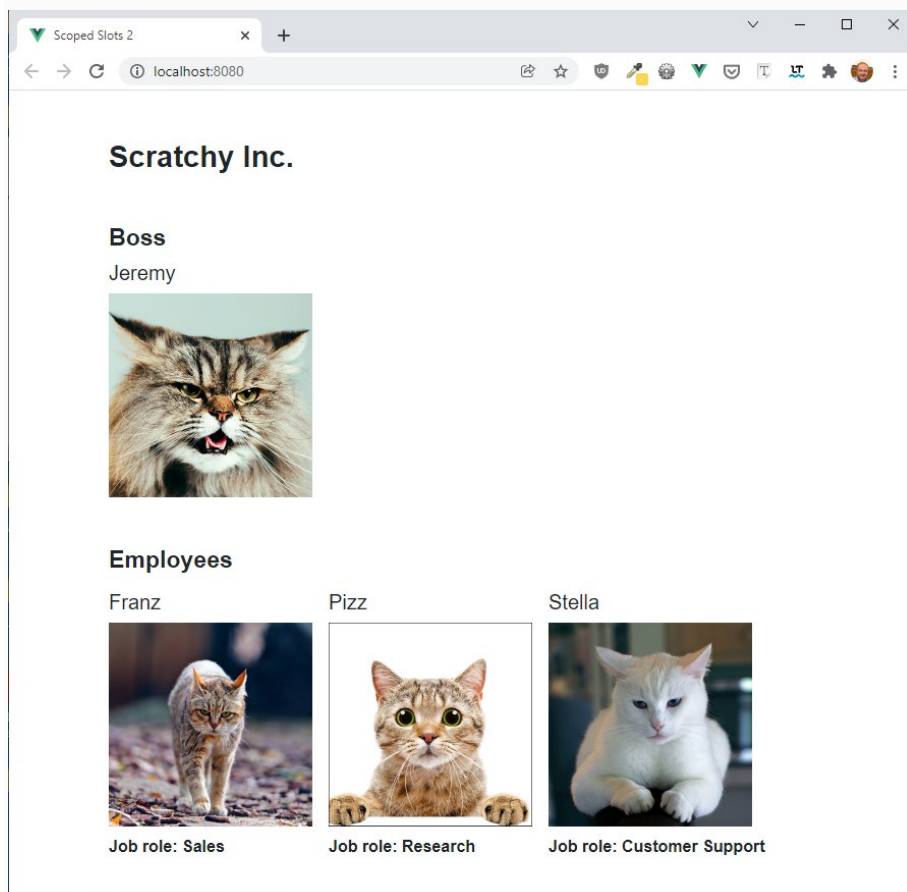
Vergleiche dazu die Vuetify Dokumentation, um eine Zeile in der Data Table zu definieren:

<https://vuetifyjs.com/en/components/data-tables/#item>

```
<template>
  <v-data-table
    :headers="headers"
    :items="desserts"
    class="elevation-1"
  >
    <template v-slot:item.calories="{ item }">
      <v-chip
        :color="getColor(item.calories)"
        dark
      >
        {{ item.calories }}
      </v-chip>
    </template>
  </v-data-table>
</template>
```

Ich hoffe, du verstehst nun wie das funktioniert!

Aufgabe 2: Ausgehend von **Slots 2 START.zip**, erstelle das Template für die Komponente **Catogramm.vue** (so nennt man ein Organigramm mit Katzen 🐱)



Robert Baumgartner

Die Parent Komponente könnte so aussehen:

```
<template>
  <div class="container mt-5" id="app">
    <Catogram :cats="cats">
      <template #boss="{item}">
        <h5>{{ item.name }}</h5>
        
      </template>
      <template #employees="{item}">
        <div>
          <h5>{{ item.name }}</h5>
          
          <p class="mt-2 fw-bold">Job role: {{ item.role }}</p>
        </div>
      </template>
    </Catogram>
  </div>
</template>
```

P.S. Schau dir auch in App.vue an, wie man Datenfiles lokal aus den Assets importiert!