

1. Einleitung

In allen bisherigen Beispielen haben wir unsere Komponenten als selbstschließende Elemente verwendet. Um Komponenten zu erstellen, die auf sinnvolle Weise zusammengesetzt werden können, müssen wir sie ineinander verschachteln können, wie wir es bei HTML-Elementen tun.

Beispiel:

```
<div id="app">
  <p>Ein Beispiel für
    <span>Verschachtelung</span>
  </p>
</div>
```

Wenn du versuchst, eine Komponente mit einem schließenden Tag zu verwenden und darin einen Inhalt zu platzieren, wirst du sehen, dass Vue diesen einfach verschluckt. Alles, was sich innerhalb der öffnenden und schließenden Tags der Komponente befindet, wird durch die gerenderte Ausgabe der Komponente selbst ersetzt. Daher haben wir bisher mittels **props** Inhalte übergeben.

AppButton.vue (ohne Slot)

```
<template>
  <div>
    <button>{{ text }}</button>
  </div>
</template>

<script>
export default {
  props: ['text'],
};
</script>
```

App.vue

```
<template>
  <div id="app">
    <AppButton text="Home"></AppButton>
  </div>
</template>
```

Es wäre es allerdings flexibler und schöner, wenn wir die **AppButton** Komponente wie ein HTML Element verwenden könnten:

App.vue

```
<template>
  <div id="app">
    <AppButton>Home</AppButton>
    <AppButton>About</AppButton>
  </div>
</template>
```

Robert Baumgartner

Mittels Slots können wir genau das machen.

Ein Slot ist ein Platzhalter innerhalb der Komponente für Inhalt, den die Komponente übergeben bekommt. Der Inhalt wird aber nicht mittels Props übergeben, sondern steht zwischen dem Begin- und Endtag der Komponente (wie oben).

Übrigens heißt dieses Feature in Angular `ng-content` und in React `children`.

AppButton.vue (mit Slot)

```
<template>
  <div>
    <button><slot /></button>
  </div>
</template>

<script>
export default {};
</script>

<style scoped>
button {
  padding: 5px;
  border: 1px solid black;
  border-radius: 10px;
  text-transform: uppercase;
  width: 150px;
  height: 185px;
  background-color: white;
}
</style>
```

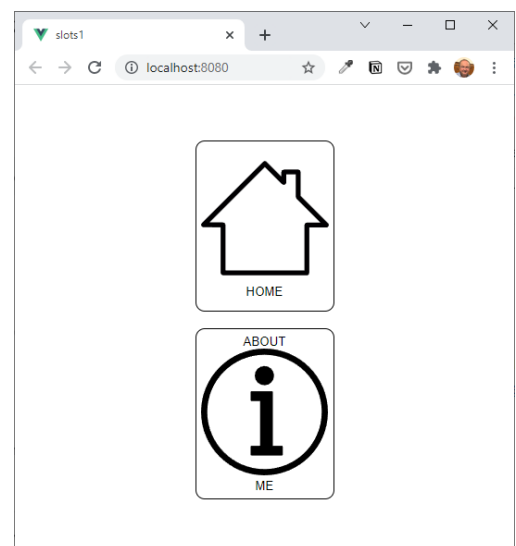
Praktisch: Der Inhalt kann auch aus mehreren Zeilen HTML Code bestehen. Die Komponente bleibt gleich.

AppButton.vue (mit Slot)

```
<template>
  <div>
    <button><slot /></button>
  </div>
</template>
```

App.vue (mit unterschiedlichem HTML Code für den Slot)

```
<template>
  <div id="app">
    <AppButton> Home</AppButton>
    <br />
    <AppButton>About  Me</AppButton>
  </div>
</template>
```



Die Komponente kann also viel flexibler verwendet werden. Um diese Flexibilität ohne Slots zu haben, müssten wir für die Komponente jede Menge Props definieren (Icon vorne, Icon hinten, eine Zeile Text, zwei Zeilen Text, etc.). Eine unschöne Sache, die als *State Explosion* bekannt ist.

Das Prinzip der Slots hat den Erstellern von Vuetify geholfen, ihre Library vielseitiger und benutzerfreundlicher zu gestalten. Dabei werden Props verwendet, um Styling zu beeinflussen und Slots um HTML Code an die Komponenten weiterzugeben. Oft enthält dieser Code selbst wieder Komponenten.

Beispiel:

```
<v-card width="500">
  <v-img :src="car.image" contain></v-img>
  <v-card-title class="text-h5 font-weight-bold d-flex justify-center mt-5">
    {{ car.title }}
    <span v-if="car.status == 'reserved'" class="ml-3"> *RESERVED* </span>
  </v-card-title>
</v-card>
```

Code für Slot von v-card

Code für Slot von v-card-title

Zurück zum Button. Wie können wir nun einen Event, der im Slot auftritt an den Benutzer des Buttons weitergeben? Die Antwort ist: `v-on="$listeners"`. Das ist eine praktische Möglichkeit, alle Events eines Elements/einer Komponente an die übergeordnete Komponente weiterzuleiten. Dadurch kann der Benutzer unserer Komponente auf alle gewünschten Events bei der Verwendung reagieren. Hier ein Beispiel:

App.vue

```
<template>
  <div id="app">
    <AppButton @click="handle"> Home</AppButton>
    <br />
    <AppButton>About  Me</AppButton>
  </div>
</template>
```

AppButton.vue

```
<template>
  <div>
    <button v-on="$listeners"><slot /></button>
  </div>
</template>
```

Du kannst aber auch direkt den Event wie gewohnt direkt beim Element abfangen.

```
<AppButton> Home</AppButton>
```

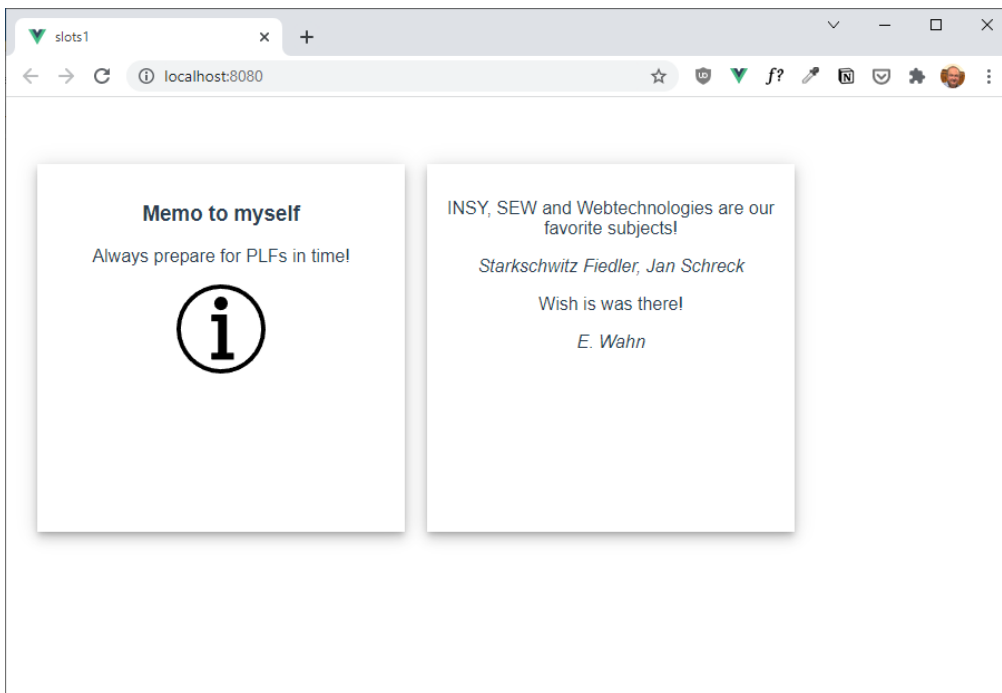
Aufgabe 1: Erstelle eine Komponente **Sheet.vue**. Die Komponente besitzt einen Slot. Die Komponente verwendet intern die Klasse **slot-style**:

```
.slot-style {
  padding: 15px;
  width: 300px;
  height: 300px;
  box-shadow: 0 3px 5px -1px rgba(0, 0, 0, 0.2),
              0 6px 10px 0 rgba(0, 0, 0, 0.14),
              0 1px 18px 0 rgba(0, 0, 0, 0.12);
}
```

Robert Baumgartner

Testcode (mit Typo von E. Wahn):

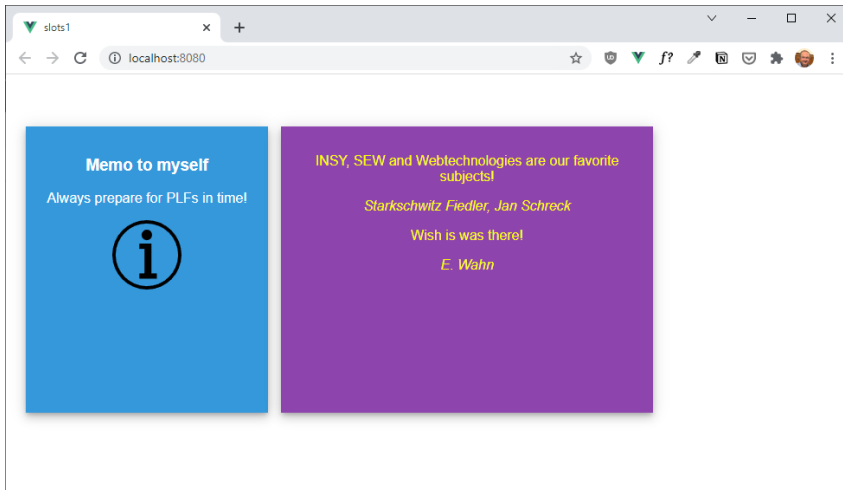
```
<div style="display:flex">
  <Sheet style="margin-left:20px">
    <h3>Memo to myself</h3>
    <p>Always prepare for PLFs in time!</p>
    
  </Sheet>
  <Sheet style="margin-left:20px">
    <p>INSY, SEW and Webtechnologies are our favorite subjects!</p>
    <span style="font-style: italic">Starkschwitz Fiedler, Jan Schreck</span>
    <p>Wish i was there!</p>
    <span style="font-style: italic">E. Wahn</span>
  </Sheet>
</div>
```



Pro Aufgabe: Die Textfarbe, Hintergrundfarbe, Weite und Höhe sollen als Props von Sheet übergeben werden. Margin-left ist immer fix 15px.

```
<div style="display:flex">
  <Sheet bg-color="#3498db" color="white" width="250" height="300">
    <h3>Memo to myself</h3>
    <p>Always prepare for PLFs in time!</p>
    
  </Sheet>
  <Sheet bg-color="#8e44ad" color="yellow" width="400" height="300">
    <p>INSY, SEW and Webtechnologies are our favorite subjects!</p>
    <span style="font-style: italic">Starkschwitz Fiedler, Jan Schreck</span>
    <p>Wish is was there!</p>
    <span style="font-style: italic">E. Wahn</span>
  </Sheet>
</div>
```

Robert Baumgartner



Named Slots

Ein Slot ist eine feine Sache, doch wie sieht es aus, wenn man mehr als einen Slot benötigt? Dann kann man den Slots Namen geben und sich dann bei der Verwendung der Komponente darauf beziehen.

Als Beispiel definieren wir eine **AppLayout** Komponente mit drei Slots:

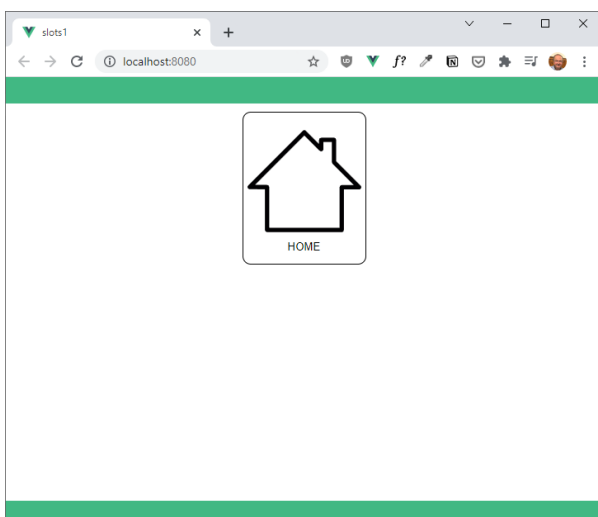
```
<div class="layout">
  <header><slot name="header"></slot></header>
  <main><slot></slot></main>
  <footer><slot name="footer"></slot></footer>
</div>
```

Diese können wir in **App.vue** wie folgt verwenden:

```
<div id="app">
  <AppLayout>
    <AppButton> Home</AppButton>
  </AppLayout>
</div>
```

Da wir den Namen des Slots nicht angegeben haben, kommt der HTML Code in den unbenannten Slot (Default Slot).

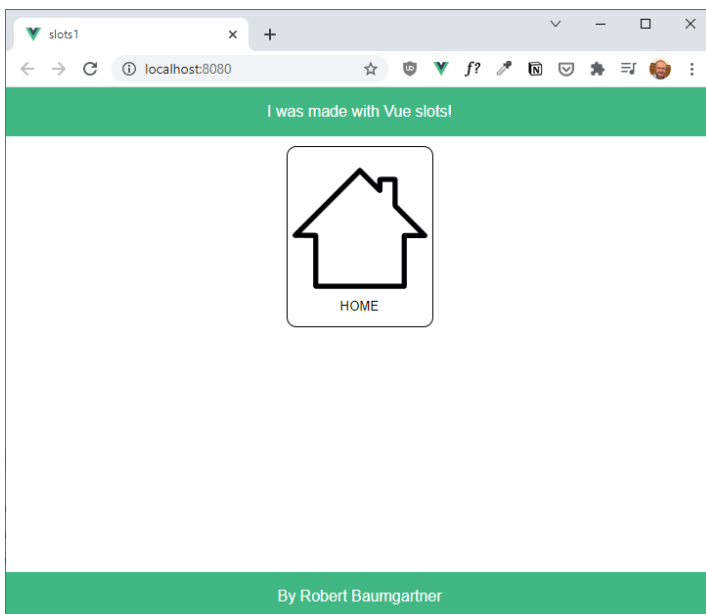
Ausgabe (habe noch ein paar Styles hinzugefügt, damit die Slots besser erkennbar sind):



Robert Baumgartner

Um die Named Slots anzusprechen, verwenden wir die `v-slot` Direktive in Vue. Dazu wrappen wir den HTML Code in ein `template` Tag.

```
<div id="app">
  <AppLayout>
    <template v-slot:header>
      I was made with Vue slots!
    </template>
    <AppButton> Home</AppButton>
    <template v-slot:footer>
      By Robert Baumgartner
    </template>
  </AppLayout>
</div>
```



Die Angabe des Defaultslots ist optional:

```
<div id="app">
  <AppLayout>
    <template v-slot:header>
      I was made with Vue slots!
    </template>
    <template v-slot:default>
      <AppButton> Home</AppButton>
    </template>
    <template v-slot:footer>
      By Robert Baumgartner
    </template>
  </AppLayout>
</div>
```

Wir können auch Default Werte für Slots vorgeben, die dann bei der Benutzung des Slots überschrieben werden.

Aufgabe 2: Probiere das aus! Spielt die Reihenfolge der Slots Angaben in **App.vue** eine Rolle!

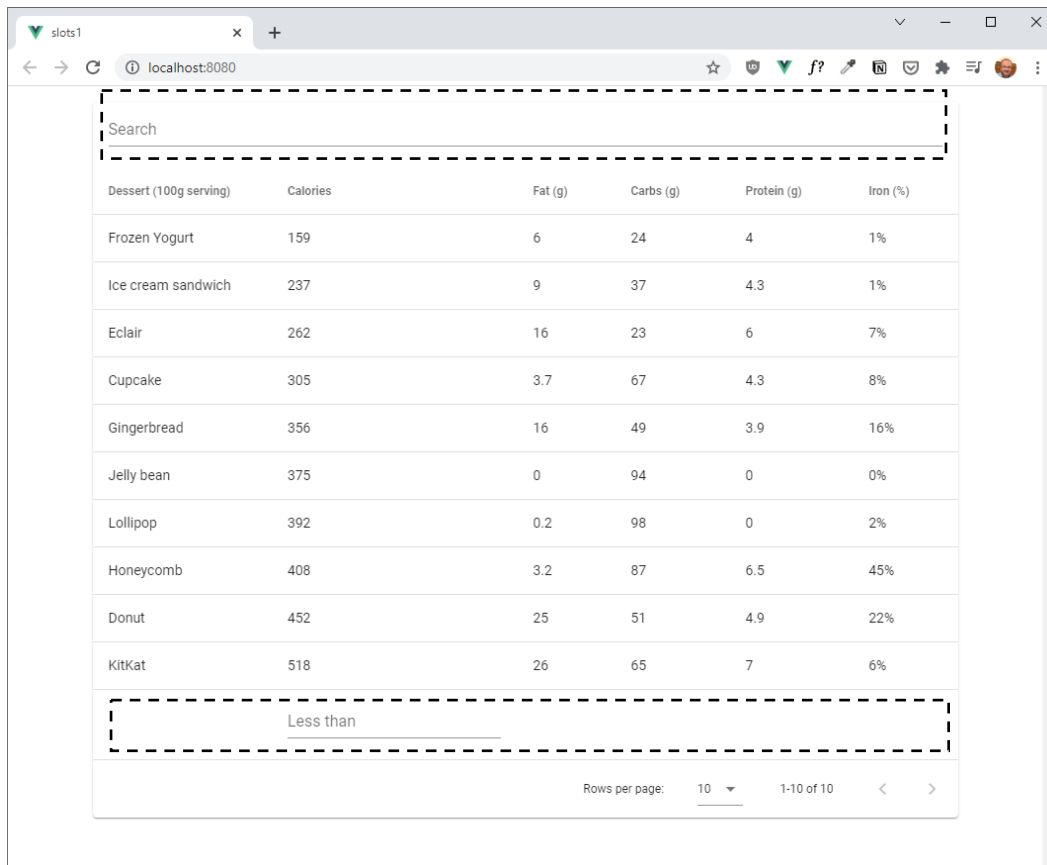
Robert Baumgartner

Beispiel aus der Vuetify Dokumentation der `v-data-table`:

```

<v-data-table
  :headers="headers"
  :items="desserts"
  item-key="name"
  class="elevation-1"
  :search="search"
  :custom-filter="filterOnlyCapsText"
>
  <template v-slot:top>
    <v-text-field
      v-model="search"
      label="Search"
      class="mx-4"
    ></v-text-field>
  </template>
  <template v-slot:body.append>
    <tr>
      <td></td>
      <td>
        <v-text-field v-model="calories" type="number" label="Less v-text-field"
      </td>
      <td colspan="4"></td>
    </tr>
  </template>
</v-data-table>

```



Dessert (100g serving)	Calories	Fat (g)	Carbs (g)	Protein (g)	Iron (%)
Frozen Yogurt	159	6	24	4	1%
Ice cream sandwich	237	9	37	4.3	1%
Eclair	262	16	23	6	7%
Cupcake	305	3.7	67	4.3	8%
Gingerbread	356	16	49	3.9	16%
Jelly bean	375	0	94	0	0%
Lollipop	392	0.2	98	0	2%
Honeycomb	408	3.2	87	6.5	45%
Donut	452	25	51	4.9	22%
KitKat	518	26	65	7	6%

Less than

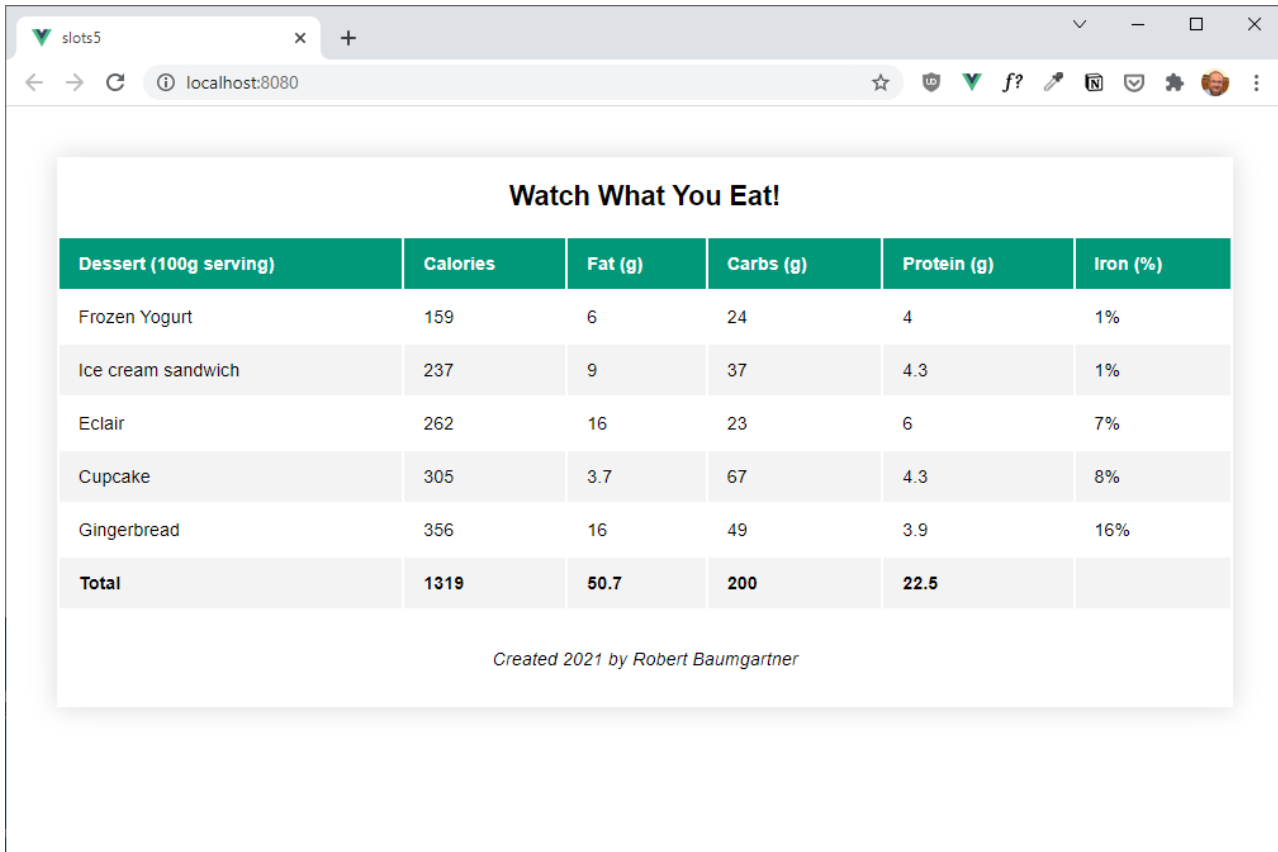
Rows per page: 10 1-10 of 10

Robert Baumgartner

Aufgabe 3: Erstelle selbst eine **DataTable** Komponente mit drei Slots: für die Überschrift (**header**), eine letzte Tabellenzeile (**body.append**) und den Footer (**footer**). Style in der **DataTable** Komponente deine Tabelle.

Teste dein Werk Hinweis: Am einfachsten ist es, wenn du die Daten von dem Vuetify Doc Beispiel nimmst und diese als Props übergibst.

Beispiel:



Dessert (100g serving)	Calories	Fat (g)	Carbs (g)	Protein (g)	Iron (%)
Frozen Yogurt	159	6	24	4	1%
Ice cream sandwich	237	9	37	4.3	1%
Eclair	262	16	23	6	7%
Cupcake	305	3.7	67	4.3	8%
Gingerbread	356	16	49	3.9	16%
Total	1319	50.7	200	22.5	

Created 2021 by Robert Baumgartner

App.vue

```
<div id="app">
  <DataTable :items="desserts" :headers="headers">
    <template v-slot:header>
      <h2>Watch What You Eat!</h2>
    </template>
    <template v-slot:footer> Created 2021 by Robert Baumgartner </template>
    <template v-slot:body.append>
      <tr class="last-row">
        <td>Total</td>
        <td>{{ totalCalories }}</td>
        <td>{{ totalFat }}</td>
        <td>{{ totalCarbs }}</td>
        <td>{{ totalProtein }}</td>
        <td></td>
      </tr>
    </template>
  </DataTable>
</div>
```


Robert Baumgartner

Die letzte Daten in der letzten Zeile in der Tabelle (`totalCalories,...`) sind computed Properties.

Hier noch meine Styles aus der **DataTable** Komponente:

```
<style scoped>
.data-table {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-content: center;
  margin: 40px;
  border-collapse: collapse;
  font-size: 0.9em;
  font-family: sans-serif;
  min-width: 400px;
  box-shadow: 0 0 20px rgba(0, 0, 0, 0.15);
}
.data-table td,
th {
  text-align: left;
}

.data-table footer {
  margin: 30px 0 30px 0;
  font-style: italic;
}

.data-table thead tr {
  background-color: #009879;
  color: #ffffff;
  text-align: left;
}

.data-table th,
.data-table td {
  padding: 12px 15px;
}

.data-table tbody tr {
  border-bottom: 1px solid #dddddd;
}

.data-table tbody tr:nth-of-type(even) {
  background-color: #f3f3f3;
}
</style>
```