Robert Baumgartner

1. Einleitung

Nachdem du schon einige Erfahrung mit Vuetify sammeln konntest, wollen wir uns nun einige grundlegende Konzepte zum Layout ansehen, mit denen du deine Designideen verwirklichen kannst.

Zwei wichtige Layout-Komponenten in Vuetify sind: `v-app` und `v-main`.

Die `v-app` Komponente ist der Einstiegspunkt (Root) deiner Anwendung und ein direkter Ersatz für `<div id="app">`.

Die `v-main` Komponente befindet sich immer innerhalb von `v-app` und stellt den Bereich dar, wo die wechselnden Inhalte deines Vuetify GUIs dargestellt werden (oft mittels `router-view`). Häufig gibt es aber Bereiche, deren Position und Inhalte fix sind (diese enthalten meist `router-links` oder zusätzliche Informationen). Das sind unter anderem Navigationsleisten (oben, unten und seitlich).

Im Arbeitsblatt *Vuetify – Navigation* werden diese Elemente behandelt. Dafür solltest du aber schon ein gutes Verständnis für Layout haben. Und genau darum geht es in diesem Arbeitsblatt. Deshalb: Erarbeite dir zuerst hier die Basiskenntnisse.

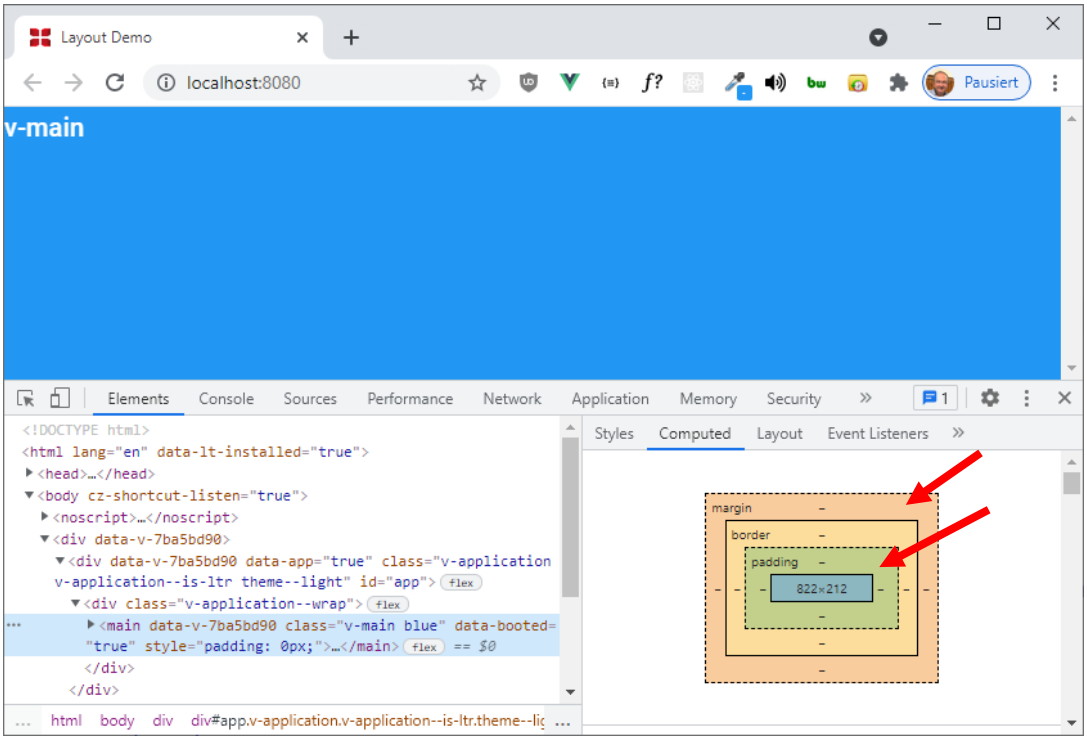
Wir werden daher das gesamte GUI in die `v-main` Komponente geben und uns die Layout Möglcihkeiten ansehen. Diese sind dann natürlich auch für die Gestaltung der Navigationskomponenten wie `v-app-bar`, `v-navigation-drawer`, `v-footer`, etc. relevant.

2. Simples Layout

Einfachstes Beispiel:

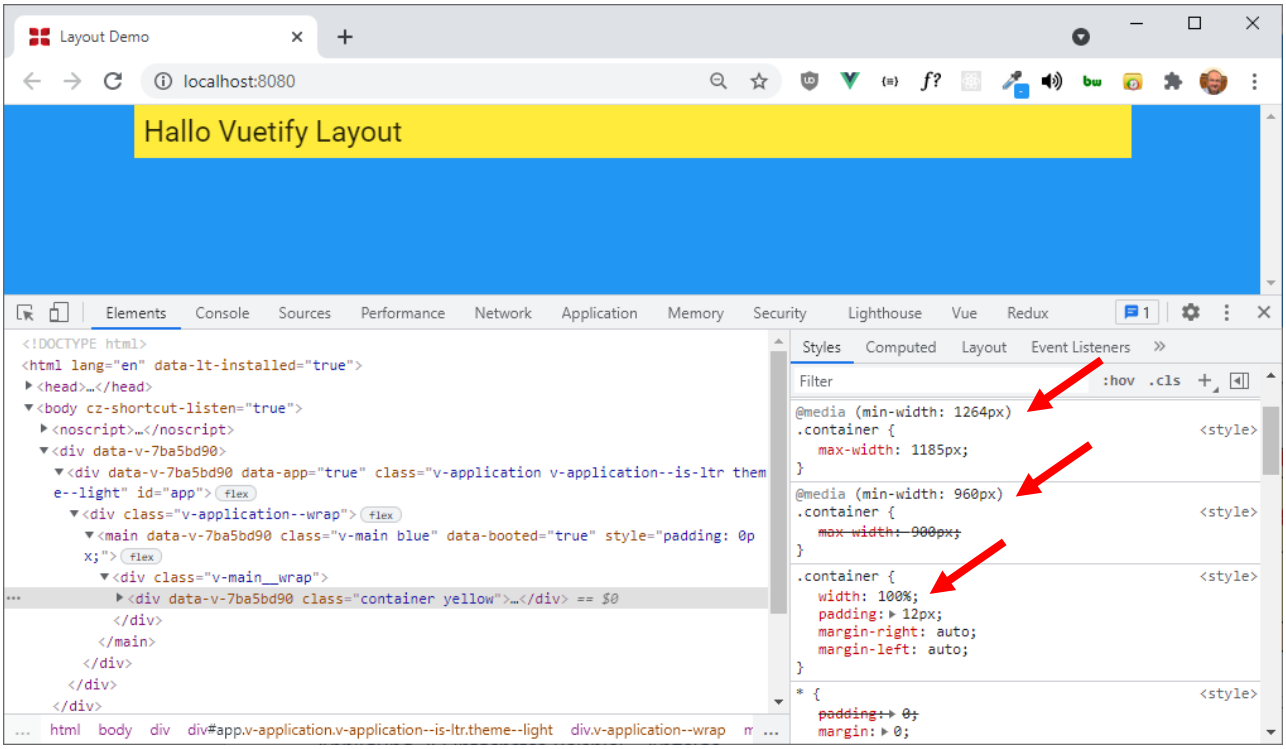
```
<v-app>
  <v-main class="blue">
    <span class="white--text title">v-main</span>
  </v-main>
</v-app>
```

`v-main` hat standardmäßig keine Margin und kein Padding (siehe rote Pfeile). Dem zugrunde liegt die Überlegung, dass der Designer, die Designerin über den gesamten Platz verfügen soll. Daher ist die Höhe auch 100%, wie du an der blauen Fläche in der Abbildung unten siehst.



Häufig aber möchte man Seitenränder haben, die sich auch an die Auflösung anpassen. Diesen dynamischen Abstand kannst du mit der `v-container` Komponente erzeugen. Somit ist die Hierachie `v-app`, `v-main` und `v-container` der Standardfall für ein Layout.

```
<v-app>
  <v-main class="blue">
    <v-container yellow ><h3 class="display-1">Hallo Vuetify Layout</h3></v-container>
  </v-main>
</v-app>
```



In der obigen Abbildung siehst du, dass die Weite des Containers von der Bildschirmauflösung abhängig ist (rote Pfeile). Außerdem gibt es ein Padding von 12px nach allen Richtungen. Der `v-container` wird in der Horizontalen mit `margin-right:auto` und `margin-left:auto` zentriert. Das funktioniert, da der Container im Prinzip ein `div` ist und ein `div` ein Block-Element ist (erinnere dich bitte, dass Inline-Elemente nicht mit `auto` zentriert werden können).

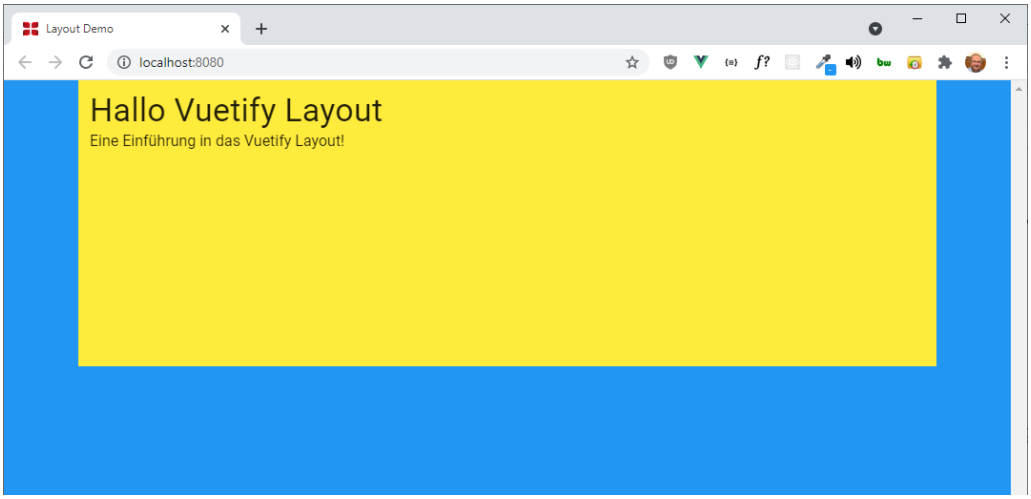
Aufgabe 1: Was passiert, wenn ein `v-container` einen `v-container` enthält? Welche Margin, welches Padding findet Verwendung?

Der `v-container` hat ein optionales Property `fluid` welches die Abstände von den Seitenrändern des Containers entfernt, sodass er dann 100% der Weite des Browserbereiches hat. Das Padding von 12px bleibt. Das kann praktisch sein, wenn du die gesamte Breite des Browserfensters benutzen möchtest.

Die Höhe des Containers ergibt sich, wenn nichts anderes angegeben wird, aus den im Container enthaltenen Elementen.

Du kannst einem Container aber auch eine bestimmte Höhe mit einem CSS Style geben:

```
<v-main class="blue">
  <v-container yellow style="height:300px">
    <h3 class="display-1">Hallo Vuetify Layout</h3>
    <p>Eine Einführung in das Vuetify Layout!</p>
  </v-container>
</v-main>
```



3. Flexbox

Sobald du einen `v-container` hast, kannst du den Inhalt auch mittels Flexbox oder Grid anordnen. Das erlaubt dir auf elegante Weise häufige Layout-Probleme zu lösen.

Kurze Wiederholung: Das Flexbox-Layout zielt darauf ab Elemente in einem Container anzuordnen, auszurichten und den Platz zwischen ihnen zu verteilen, auch wenn ihre Größe unbekannt und/oder dynamisch ist. Das ist eben der Unterschied zu einem Grid: Dort ist die Größe der Elemente von vornherein bekannt (Anzahl Spalten und deren Breite).

Flexbox ist keine einzelne Eigenschaft - es ist ein ganzes System. Es umfasst Eigenschaften für das übergeordnete Element, bekannt als Flex-Container, und für die untergeordneten Elemente, bekannt als Flexelemente.

Die Hauptidee hinter dem Flex-Layout ist es, dem Container die Möglichkeit zu geben, die Breite/Höhe (und die Reihenfolge) seiner Elemente zu verändern, um den verfügbaren Platz bestmöglich auszufüllen (vor allem, um sich an alle Arten von Anzeigegeräten und Bildschirmgrößen anzupassen). Ein Flex-Container dehnt Flex-Elemente und Leerraum aus, um den verfügbaren freien Platz zu füllen oder schrumpft sie, um ein Überlaufen zu verhindern.

Wenn du dich nicht mehr genau erinnern kannst, lies dir

https://css-tricks.com/snippets/css/a-guide-to-flexbox/#flexbox-background

durch und spiele mit:

https://flexbox.netlify.app/

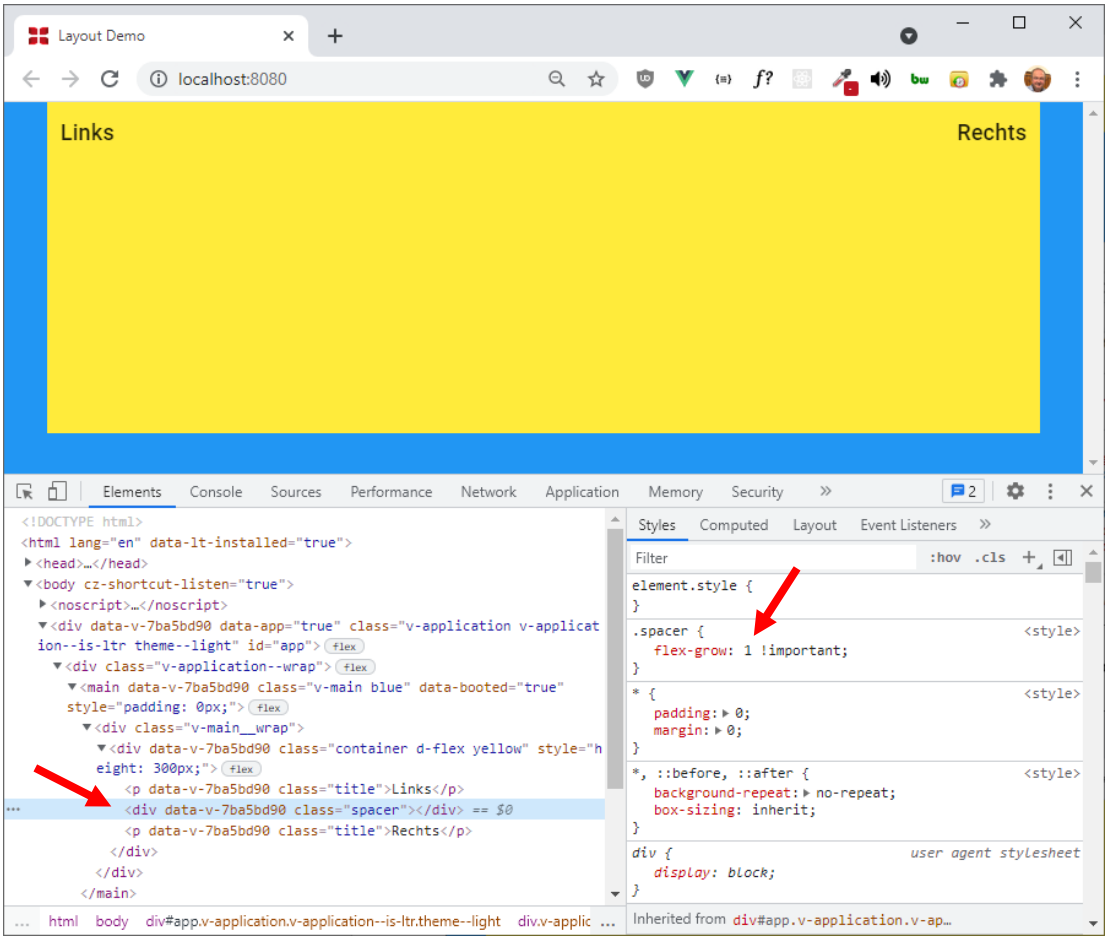
Ok, und wie hat Vuetify das umgesetzt? Wenn du Vuetify verwendest, musst du, um Flexbox zu benutzen, keine CSS Regeln wie `display:flex`, etc. verwenden. Du kannst einfach die Flex Helper verwenden. Siehe: https://vuetifyjs.com/en/styles/flex/

Beim `v-container` können die Flex Helper direkt als Property angegeben werden (via `class` ginge es natürlich auch). Bei einem `div` müssen sie als CSS Klasse angegeben werden.

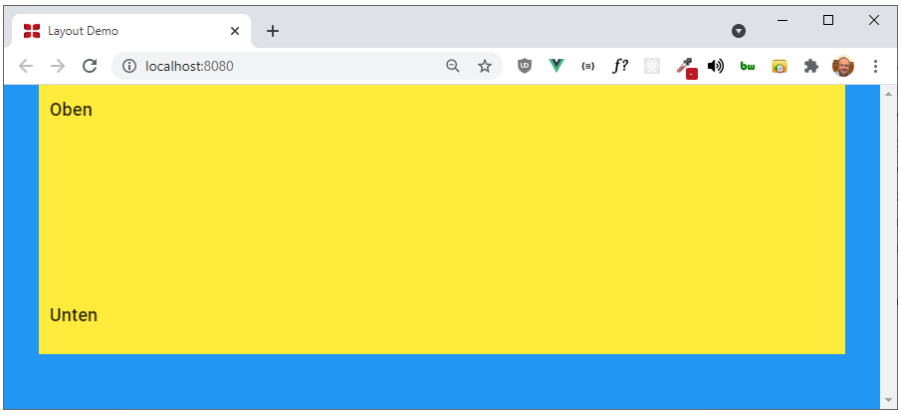
Layout Problem 1: Du hast zwei Elemente und möchtest das eine am linken Rand und das andere am rechten Rand platzieren. Um die Elemente auseinanderzupushen, kannst du `v-spacer` verwenden. Das ist im Prinzip ein `div` mit `flex-grow` (siehe rote Pfeile).

```
<v-main class="blue">
  <v-container d-flex yellow>
    <p>Links</p>
    <v-spacer></v-spacer>
    <p>Rechts</p>
  </v-container>
</v-main>
```

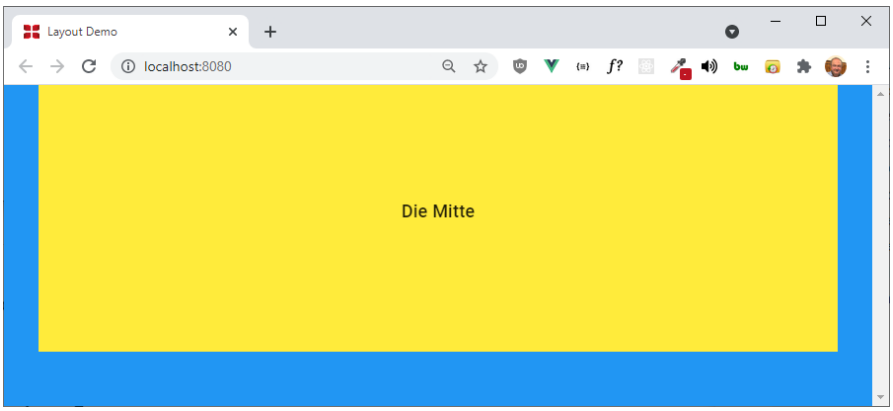
Robert Baumgartner



Aufgabe 2: Wie sieht der Code aus, um folgendes Layout Problem zu lösen?



Aufgabe 3: Wie sieht der Code aus, um folgendes Layout Problem zu lösen?



Layout Problem 2: Du hast eine Reihe von **divs** mit unterschiedlichen Inhalten, willst aber, dass die Buttons immer ganz unten sind. Der äußere Flex-Container sorgt dafür, dass die inneren Elemente gleiche Höhe haben. Der innere Flex-Container ordnet die inneren Elemente als Flex-Elemente mit Primärachse **flex-column** an. **v-spacer** pusht sie maximal auseinander.

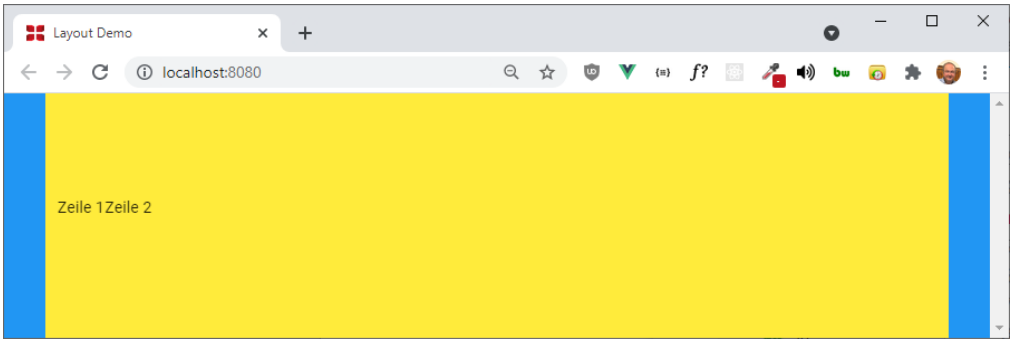
```
<v-container d-flex yellow>
  <div class="d-flex flex-column green lighten-3 pa-2 ma-2">
    <p>Top Zeile</p>
    <p>Zeile 2</p>
    <p>Zeile 3</p>
    <v-spacer></v-spacer>
    <v-btn color="primary">Next</v-btn>
  </div>
  <div class="d-flex flex-column green lighten-3 pa-2 ma-2">
    <p>Top Zeile</p>
    <v-spacer></v-spacer>
    <v-btn color="primary">Next</v-btn>
  </div>
  <div class="d-flex flex-column green lighten-3 pa-2 ma-2">
    <v-spacer></v-spacer>
    <v-btn color="primary">Next</v-btn>
  </div>
</v-container>
```

Layout Problem 3: Der **v-container** soll die ganze Höhe des verfügbaren Platzes ausfüllen.

```
<v-container style="height:100%" yellow>
<p>Zeile 1</p>
<p>Zeile 2</p>
</v-container>
```

Das Property **fill-height** des **v-containers** macht das Gleiche, legt aber zusätzlich noch andere Eigenschaften des **v-containers** fest.

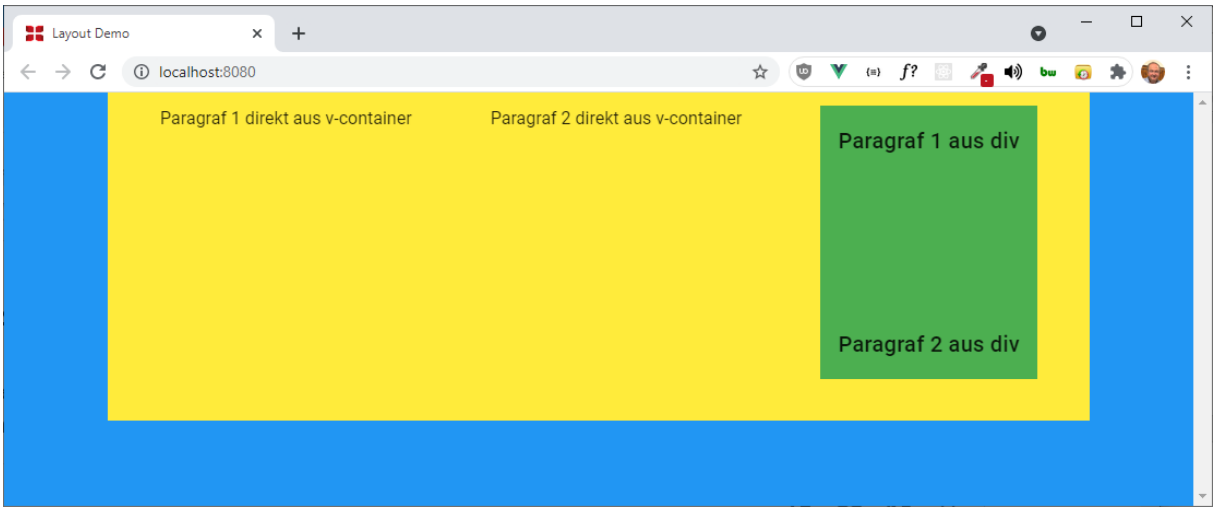
Robert Baumgartner



Aufgabe 4: Welche Eigenschaften wurden zusätzlich durch `fill-height` dem `v-container` gegeben?

In der nächsten Abbildung kannst du sehen, dass unterschiedlichste Elemente wie `span, p, h2` durch die Angabe von `d-flex` im Parent Element zu Flex-Elementen werden.

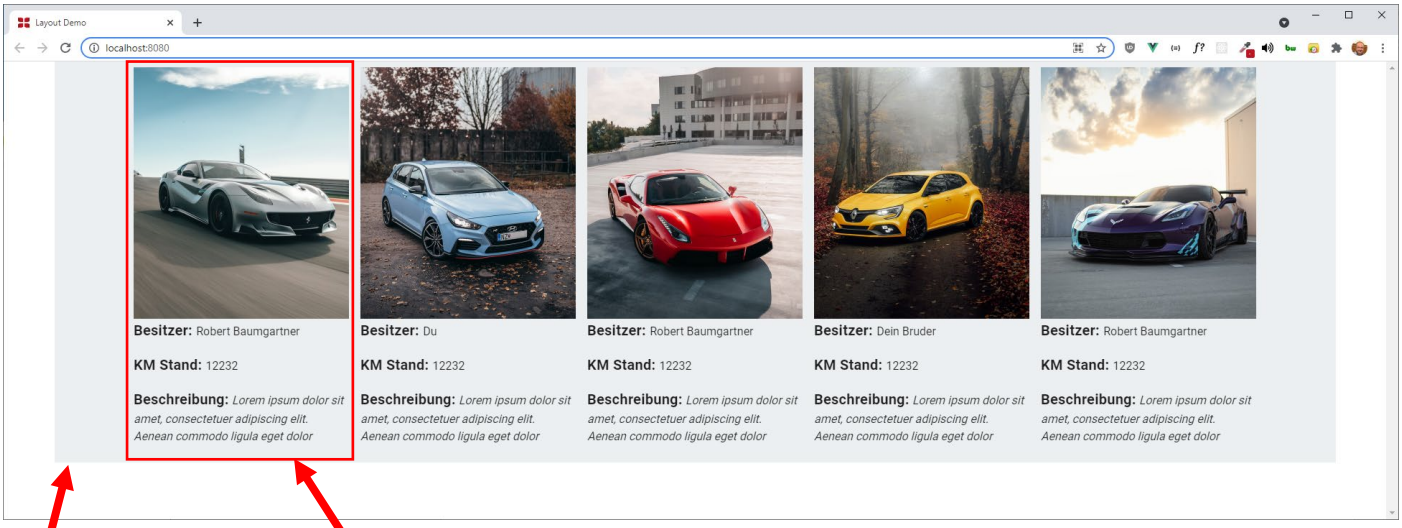
```
<v-main>
<v-container style="height:300px" yellow d-flex flex-row justify-space-around>
  <span>Paragraf 1 direkt aus v-container</span>
  <p>Paragraf 2 direkt aus v-container</p>
  <div style="height:250px" class="green pa-4 d-flex flex-column justify-space-between">
    <h2 class="title">Paragraf 1 aus div</h2>
    <span class="title">Paragraf 2 aus div</span>
  </div>
</v-container>
</v-main>
```



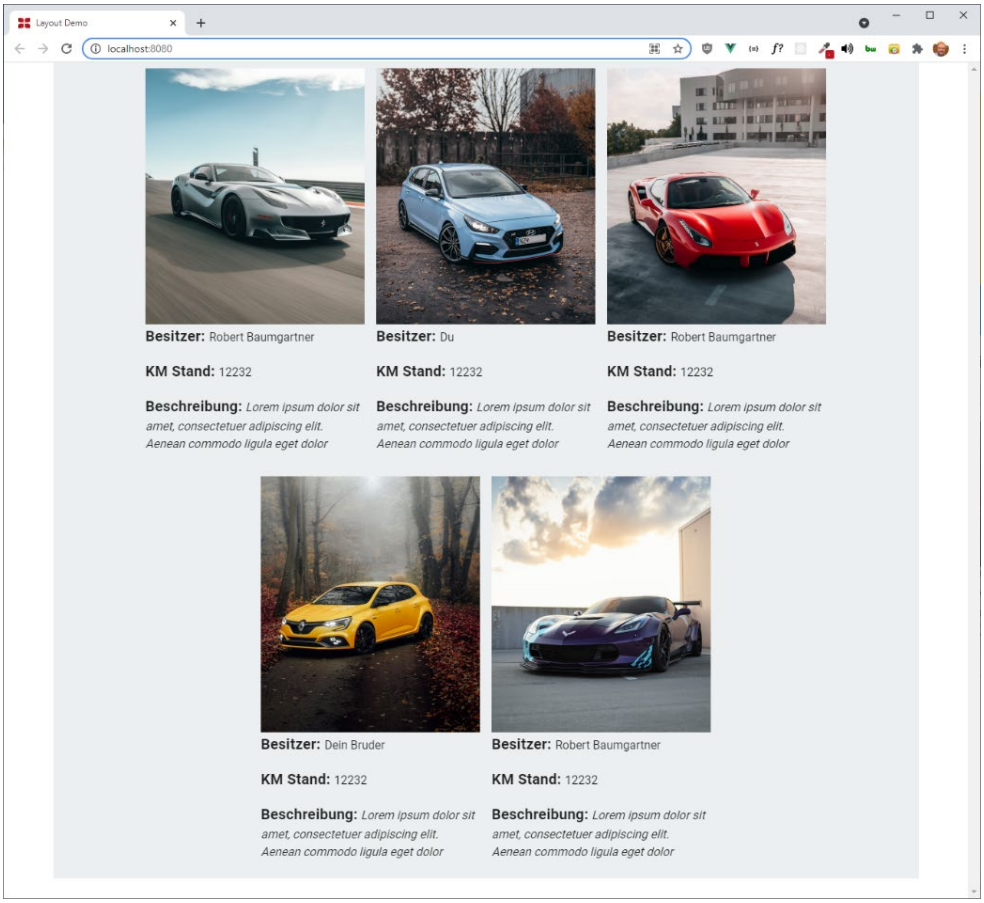
Aufgabe 5: Wie hoch ist das `div` mit dem Text "Paragraf 2 direkt aus v-container" und warum?

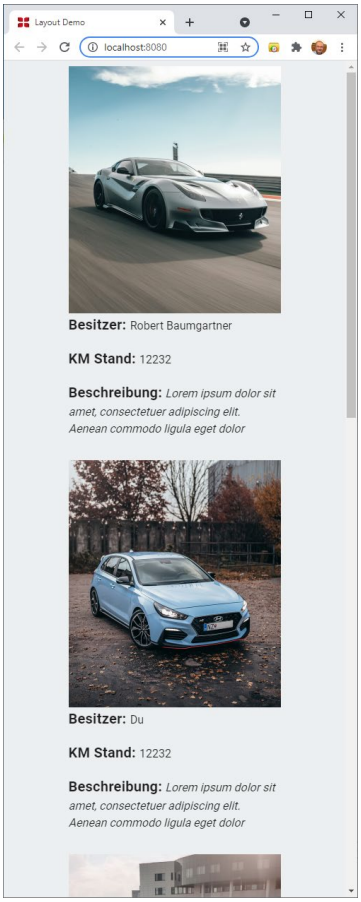
Aufgabe 6: Erstelle das Vuetify Layout, dargestellt auf der nächsten Seite (Daten und Bilder in **Vuetify Layout Angabe.zip**).

Verwende dazu zwei Flexboxen. Das GUI soll sich beim Ändern der Fenstergröße des Browsers wie gezeigt verhalten. Natürlich könntest du auch die Komponente `v-card` verwenden. Zur Übung ist es aber gut, mal alles selbst zu erstellen! Beachte, dass sich die Bilder durch die Flexbox automatisch in der Höhe anpassen!



Äußere Flexbox Innere Flexbox





4. Grid

Die Komponenten des Vuetify Grid System sind: `v-container`, `v-row`, `v-col` und `v-spacer`.

`v-row` ist eine Wrapper-Komponente für `v-col`. Sie nutzt die Flex-Eigenschaften, um das Layout und die Anordnung der mit `v-col` definierten Spalten zu kontrollieren.

`v-col` muss ein direktes Kind von `v-row` sein! Da ein Grid 12 Einheitsspalten hat, kannst du mit `v-col` definieren, wie viele Einheitsspalten deine mit `v-col` definierte Spalte breit sein soll. Das kann abhängig von der Bildschirmauflösung (**xs, sm, sm, lg, xl**) erfolgen.

Beispiel: Die mit `v-col` definierte Spalte soll immer 12 Einheitsspalten breit sein, **ab** Auflösung `md` wollen wir 6 Spalten Breite und **ab xl** 3 Spalten Breite:

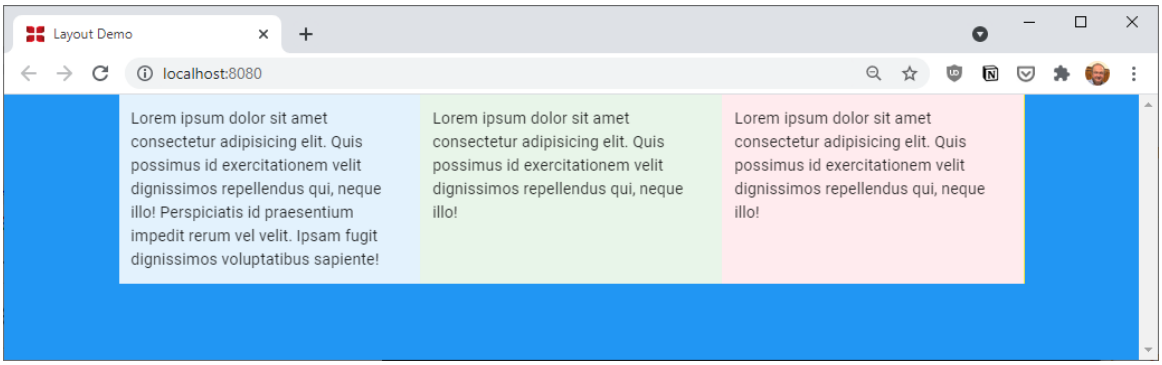
```
<v-col cols="12" md="6" xl="3" class="blue lighten-5">
```

`v-row` und `v-col` bieten verschiedene optionale Properties, um Inhalte anzuordnen (align, justify, order, offset,...). Davon später.

Da `v-row` eine Flexbox ist, kann `v-spacer` auch hier verwendet werden.

Du kannst mehr Informationen hier finden: https://vuetifyjs.com/en/components/grids/

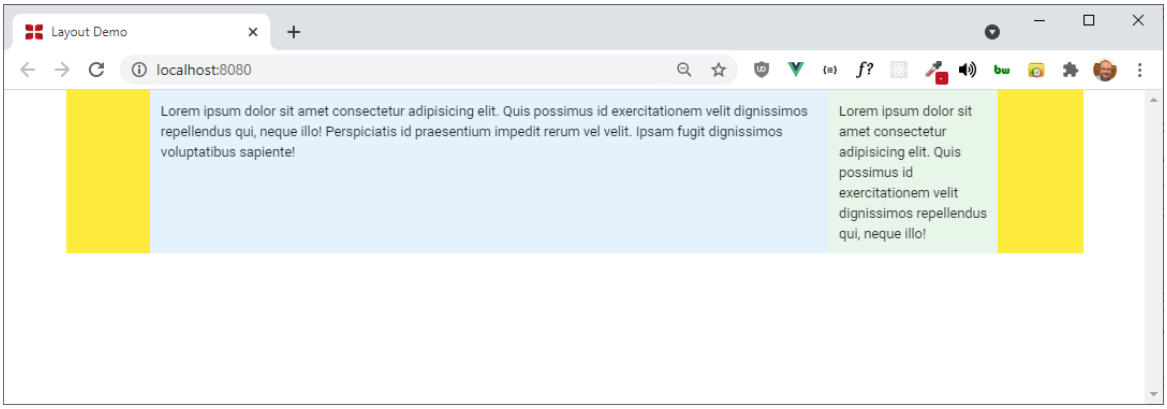
Aufgabe 7: Erstelle ein Grid mit einer Zeile und drei Spalten. Jede Spalte ist in den Auflösungen **sm, md, lg, xl** 4 Einheiten breit und in den Auflösungen **xs** 12 Einheiten breit. Verwende auch `v-container`. Teste dein Werk!

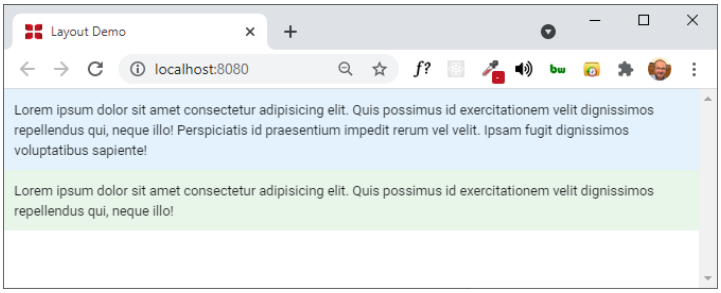


Zum Positionieren deiner Spalten, kannst du auch Einheitsspalten überspringen. Dazu kannst du das Property `offset` verwenden, das es auch für verschiedene Auflösungen gibt.

Beispiel:

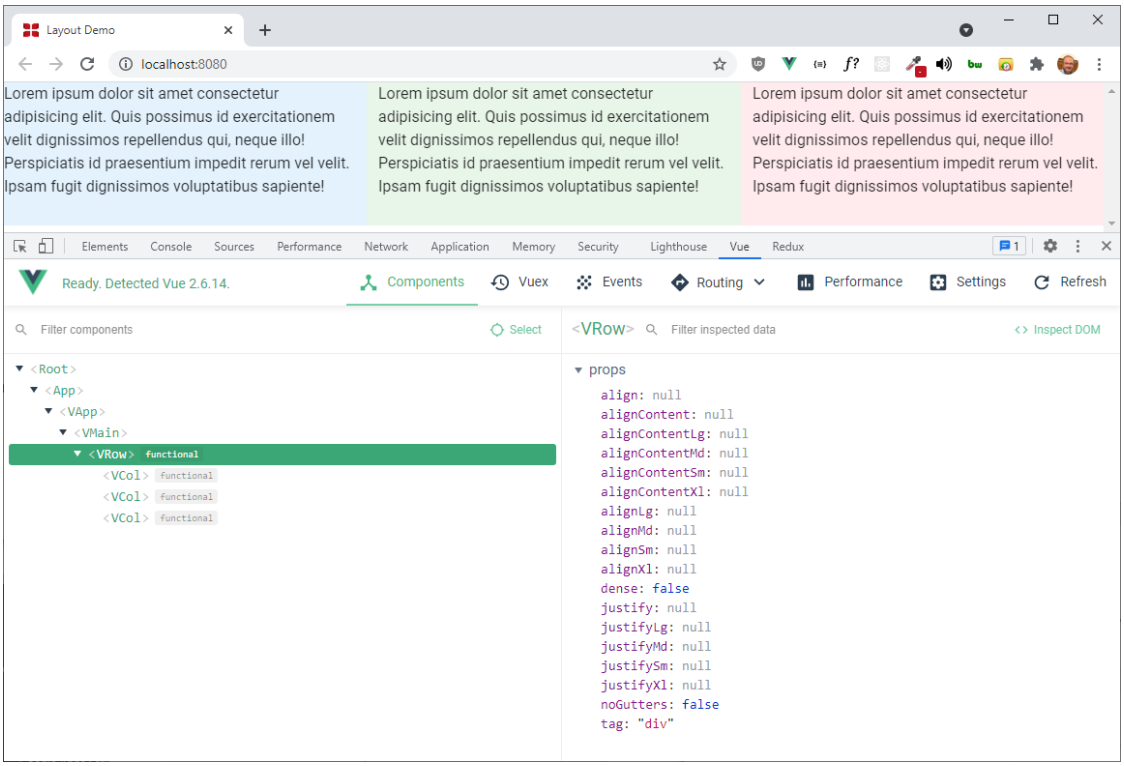
```
<v-main>
  <v-container>
    <v-row class="yellow" >
      <v-col cols="12" offset-md="1"  md="8" class="blue lighten-5">
        <span>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Quis possimus id
          exercitationem velit dignissimos repellendus qui, neque illo! Perspiciatis id
          praesentium impedit rerum vel velit. Ipsam fugit dignissimos voluptatibus sapiente!
        </span>
      </v-col>
      <v-col cols="12" md="2" class="green lighten-5">
        <span>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Quis possimus id
          exercitationem velit dignissimos repellendus qui, neque illo!
        </span>
      </v-col>
    </v-row>
  </v-container>
</v-main>
```





Ich hoffe, du hast gesehen: Die 12 Einheitsspalten stellen einen praktischen Raster für die Positionierung der eigenen Spalten dar. Allerdings kann man diesen auch ignorieren und stattdessen Flexbox Eigenschaften werden. Dazu bietet `v-row` vor allem zwei Properties an: `justify` (für die horizontale Ausrichtung) und `align` (für die vertikale Ausrichtung innerhalb einer Row). Auch diese unterstützen verschiedene Auflösungen!

Diese Properties können wir uns mittels der **Vue.js devtools** ansehen (Browser Erweiterung).

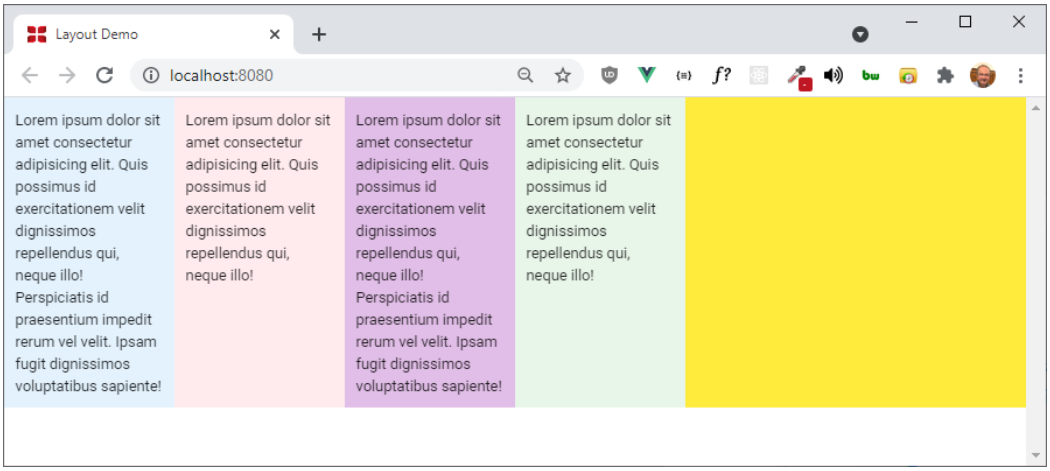


Beachte, dass im Gegensatz zu einer vollwertigen Flexbox, die primäre Achse der `v-row` immer horizontal ist! Klar, sie ist ja auch eine Row! Somit kannst du nur mit `justify` und `align` (und `align-content`) die Position der Columns beeinflussen. Allerdings kannst du immer andere Flexbox Eigenschaften als Klasse übergeben. Zum Beispiel `flex-column`. Das ist aber wenig sinnvoll, da du in dem Fall gleich einen `v-container` hättest verwenden können. Mit einem Hammer rührt man auch nicht die Suppe um.

Nehmen wir mal folgenden Code an:

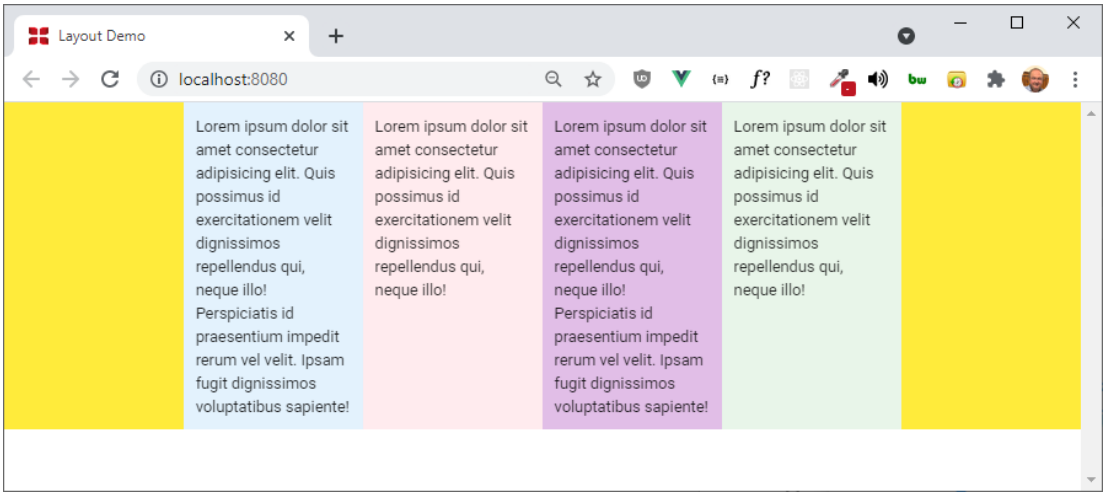
```
<v-main>
  <v-container fluid>
    <v-row class="yellow" >
      <v-col cols="12" md="2" class="blue lighten-5">
        <span>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Quis possimus id
          exercitationem velit dignissimos repellendus qui, neque illo! Perspiciatis id
          praesentium impedit rerum vel velit. Ipsam fugit dignissimos voluptatibus sapiente!
        </span>
      </v-col>
      <v-col cols="12" md="2" class="red lighten-5">
        <span>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Quis possimus id
          exercitationem velit dignissimos repellendus qui, neque illo!
        </span>
      </v-col>
      <v-col cols="12" md="2" class="purple lighten-4">
        <span>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Quis possimus id
          exercitationem velit dignissimos repellendus qui, neque illo! Perspiciatis id
          praesentium impedit rerum vel velit. Ipsam fugit dignissimos voluptatibus sapiente!
        </span>
      </v-col>
      <v-col cols="12" md="2" class="green lighten-5">
        <span>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Quis possimus id
          exercitationem velit dignissimos repellendus qui, neque illo!
        </span>
      </v-col>
    </v-row>
  </v-container>
</v-main>
```

Da die noch 4 Einheitsspalten überbleiben, verwendet **`v-row`** die Defaulteigenschaft **`justify="start".`**



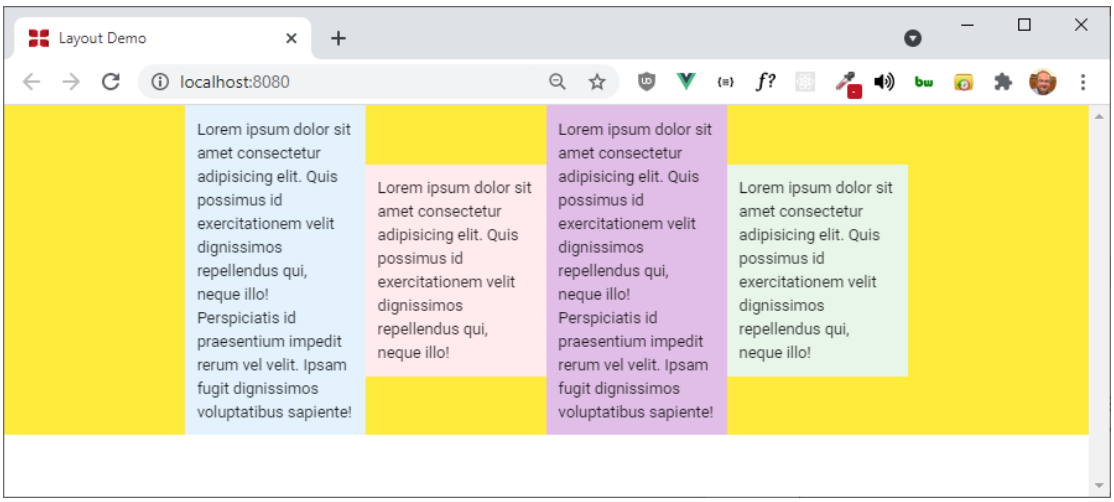
`justify="center`" bewirkt, dass die **`v-cols`** (unabhängig von Einheitspalten) in der Mitte horizontal zentriert werden.

```
<v-row class="yellow" justify="center" >
```

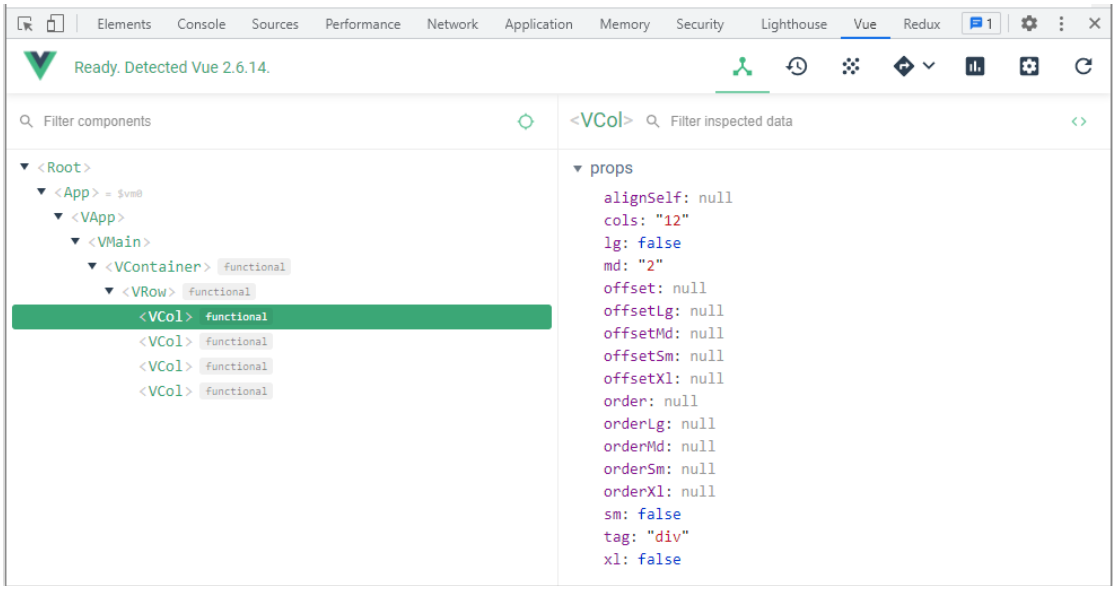


Die Eigenschaft **`align`** bezieht sich auf die Höhe der **`v-row`**. Diese richtet sich wiederum nach der längsten Spalte. Defaultmäßig haben alle Spalten **`align="stretch"`** damit sie gleich hoch sind. Du kannst nun mit **`align="center"`** die eigentliche Höhe der Spalten benutzen und diese zentrieren.

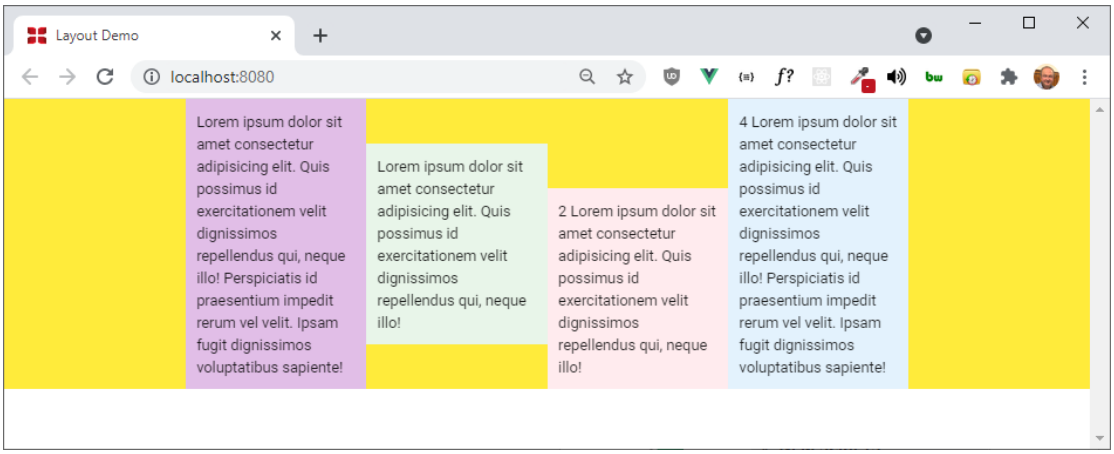
```
<v-row class="yellow" justify="center" align="center">
```



Auch v-col hat einige interessante Properties zu bieten:



Von der Flexbox ist `align-self` bekannt. Mit `order` kann du die Reihenfolge der Spalten bestimmen.



Aufgabe 8: Erstelle den Code zu obigem Bild.