



Rikkster 29 января в 23:26

# Безопасный CSS, или как писать универсальные стили

CSS\*, HTML\*

Из песочницы

Перевод

Автор оригинала: Ahmad Shadeed

Предисловие: при написании CSS-стилей, необходимо сразу учитывать, что контент страницы может быть динамическим, чтобы не возникла ситуация, где мы добавили чуть больше текста, или уменьшили ширину экрана, и вёрстка поплыла. Я наткнулся на хорошую англоязычную статью с примерами универсальных CSS-стилей для часто встречающихся ситуаций, и решил, что эта ценная информация для начинающих верстальщиков должна быть и на русском языке. Так появился этот перевод.

## Оглавление:

(кликально)

1. Умный flex-контейнер
2. Расстояние
3. Длинный контент
4. Предотвращение искажения изображения
5. Блокировка цепочки прокрутки
6. Резервное значение CSS-переменной
7. Использование фиксированной ширины или высоты
8. Отключение повторения фона
9. Вертикальные @media-запросы
10. Использование justify-content: space-between
11. Текст поверх изображений
12. Будьте осторожны с фиксированными значениями в CSS-Grid
13. Показываем полосу прокрутки только когда это необходимо
14. Ширина контента с полосой прокрутки



17. Auto Fit против Auto Fill в CSS-Grid minmax

18. Максимальная ширина IMG

19. Использование position: sticky в CSS-Grid

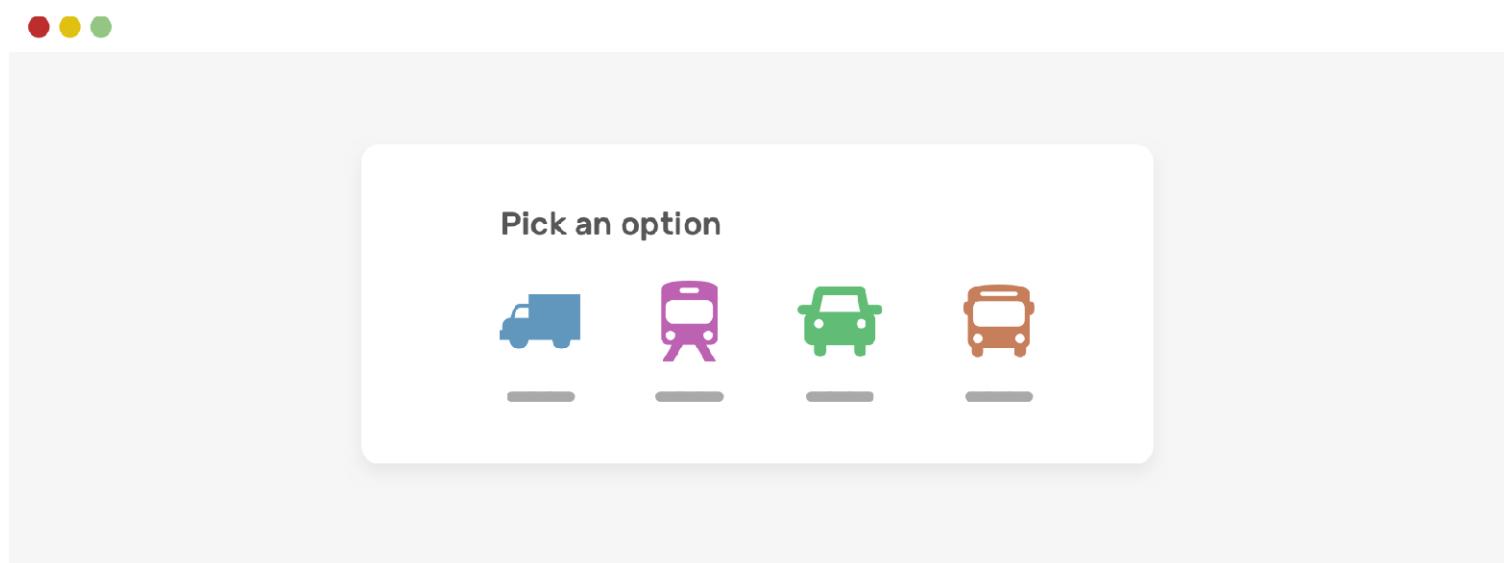
20. Группировка селекторов для разных браузеров

## Умный flex-контейнер

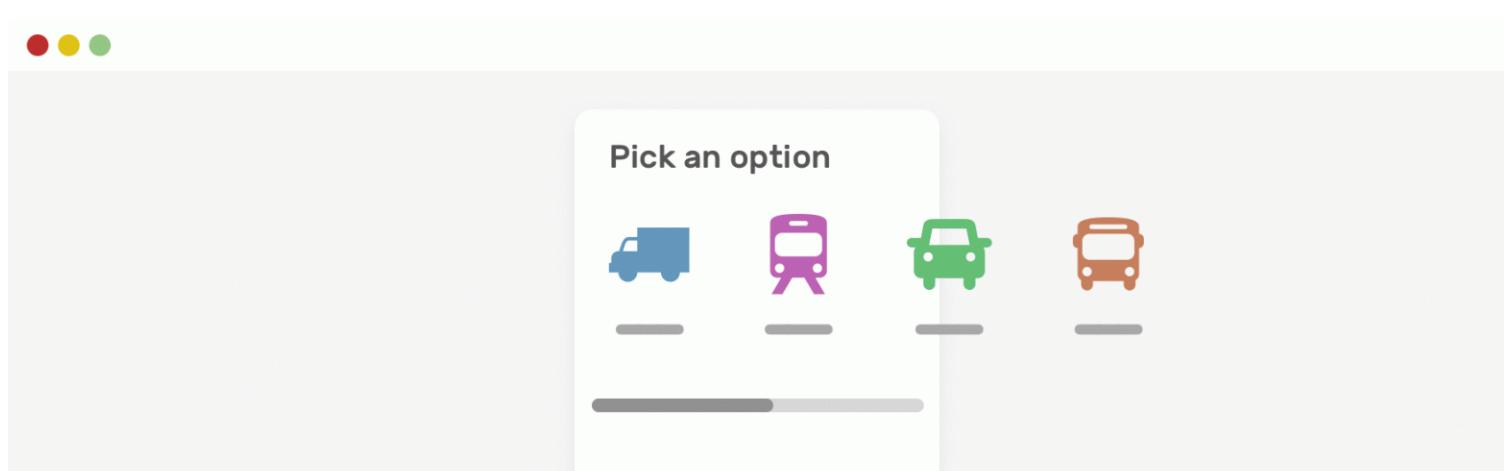
CSS flexbox — одна из самых полезных функций компоновки CSS на сегодняшний день. Очень удобно добавить `display: flex` в оболочку и расположить дочерние элементы рядом друг с другом.

Однако, когда места недостаточно, эти дочерние элементы по умолчанию не переносятся на новую строку. Нам нужно изменить это поведение с помощью `flex-wrap: wrap`

Вот типичный пример. У нас есть группа параметров, которые должны отображаться рядом друг с другом.

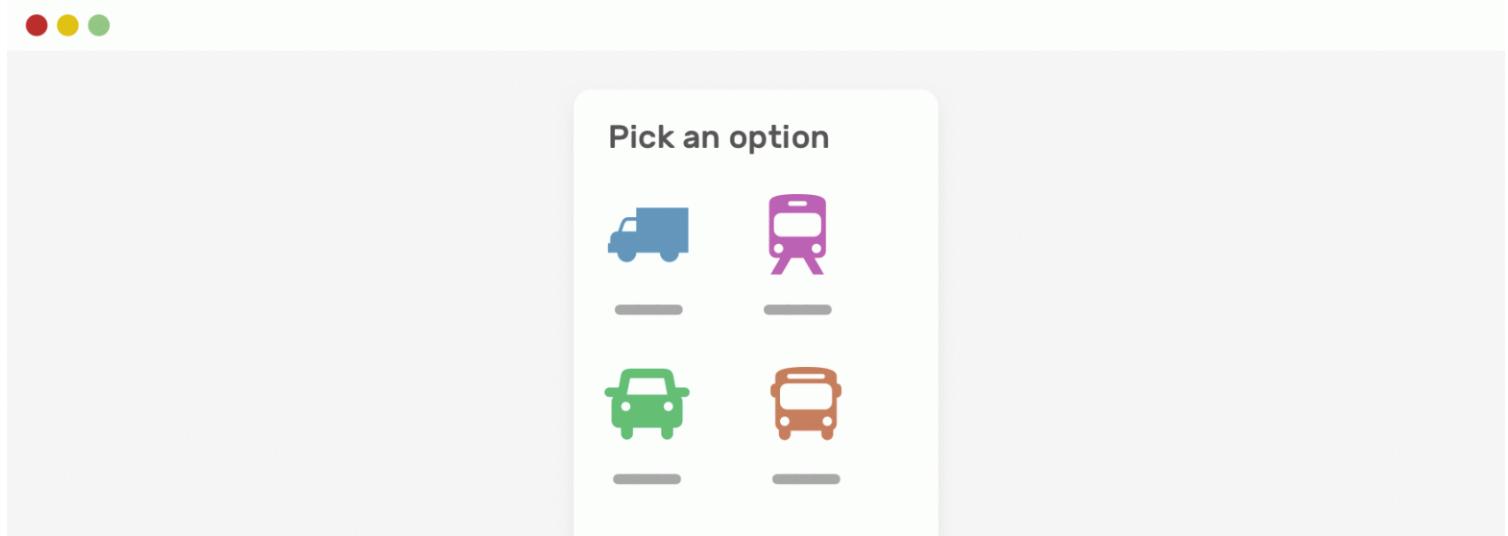


Когда контейнер меньше чем его содержимое, будет отображена горизонтальная прокрутка. Этого следует ожидать, и на самом деле это не является «проблемой».



Обратите внимание, что элементы все еще рядом друг с другом. Нам нужно чтобы элементы при уменьшении ширины контейнера переносились на другую строку, и не выходили за рамки самого контейнера. Для этого следует написать так:

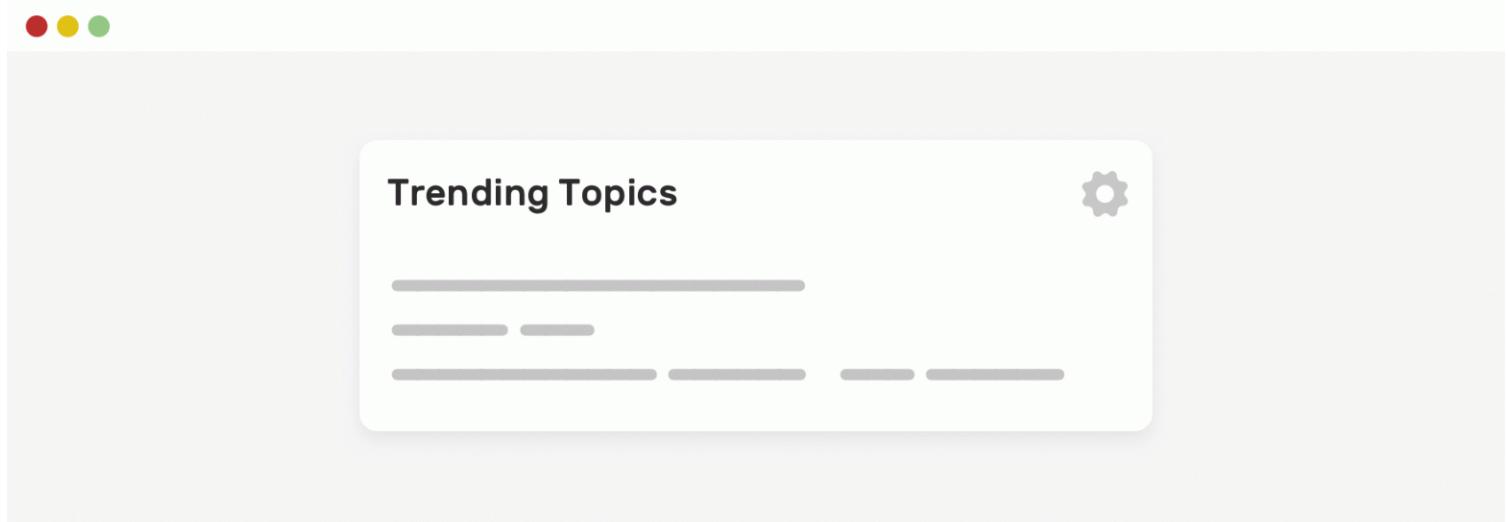
```
.options-list {  
    display: flex;  
    flex-wrap: wrap;  
}
```



отлично, теперь выглядит адекватно!

## Расстояние

Мы, верстальщики, должны учитывать разную длину контента. Это означает, что отступы должны быть добавлены к элементу даже если кажется, что они не нужны.



В этом примере у нас есть заголовок раздела и кнопка действия справа. Сейчас выглядит нормально. Но давайте посмотрим, что произойдёт, если заголовок будет длиннее:



## A list of all the useful topics for TODAY



Заметили, что текст расположен слишком близко к кнопке действия? Возможно, вы думаете о многострочном переносе, но я вернусь к этому в другом разделе. А пока сосредоточимся на отступах.



## Trending Topics



## Trending Topics with a longer title.....



```
.section_title {  
    margin-right: 1rem; /* Отступ справа */  
}
```

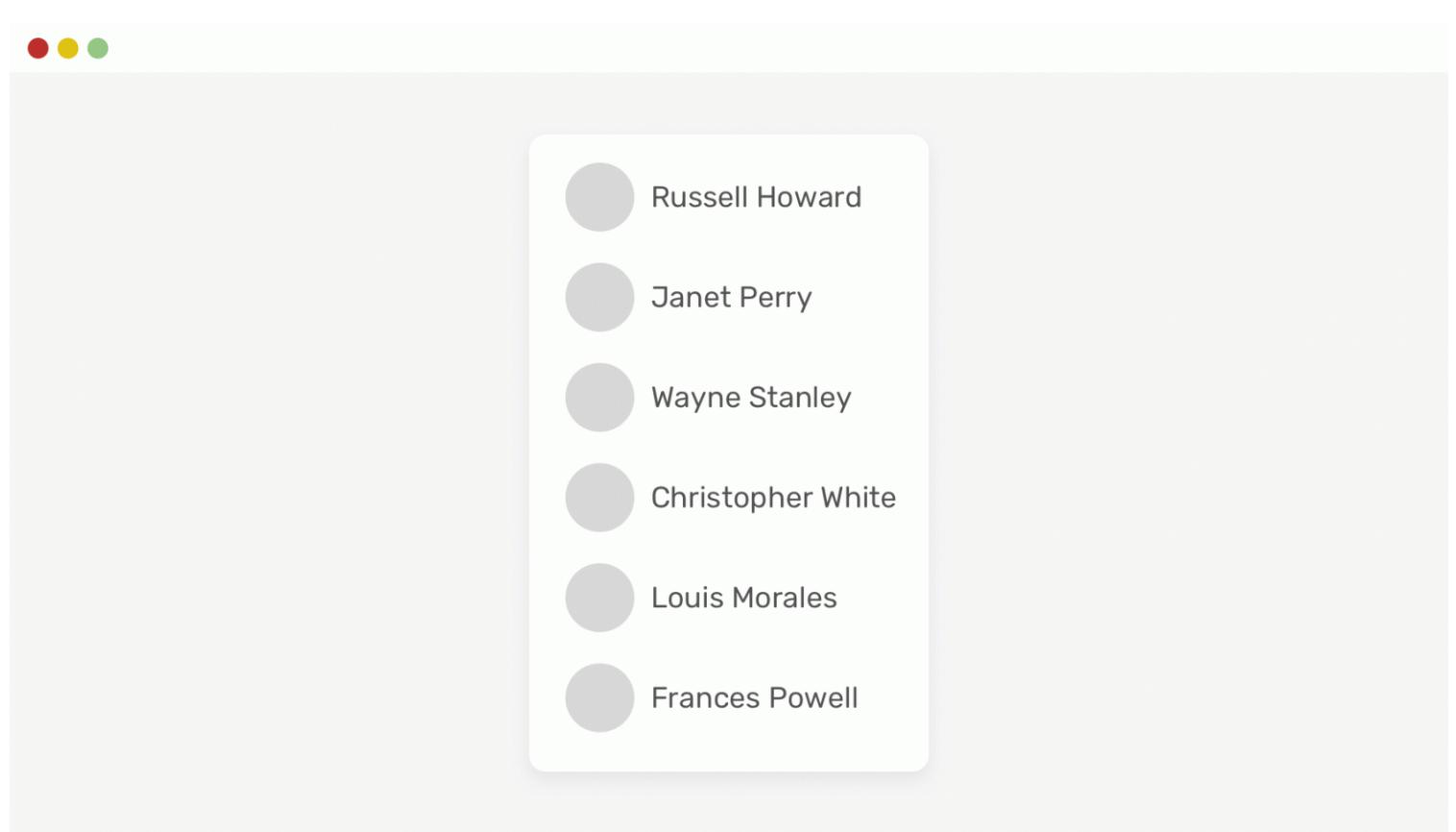
Как обрезать текст точками, читаем ниже:

**Длинный контент**

Учет длинного контента важен при построении макета. Как вы могли видеть в предыдущем примере, название раздела усекается, если оно слишком длинное. Это необязательно, но для некоторых пользовательских интерфейсов это важно учитывать.

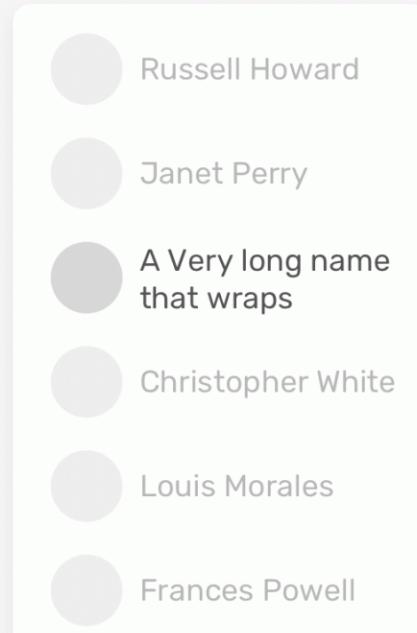
Для меня это "безопасный" подход к CSS. Приятно получить возможность исправить «проблему» до того, как она произойдет.

Вот список имен людей, и сейчас он выглядит идеально.



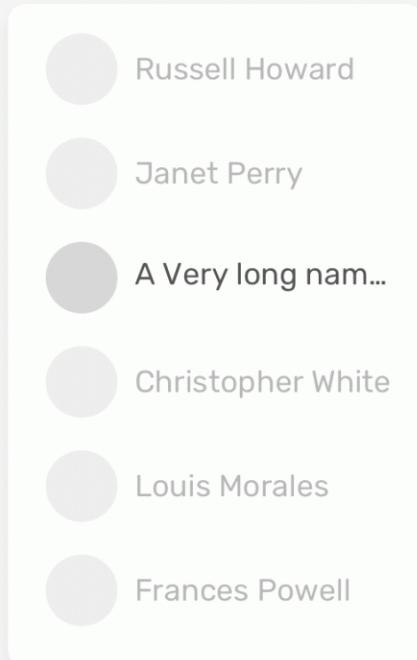
Однако поскольку это контент, созданный пользователями, нам нужно продумать стили на случай, если что-то будет слишком длинным.

См. следующий пример:



Важно, чтобы элементы выглядели однотипно. Для этого мы должны запретить перенос слов, и обрезать часть слов, если они выходят за ширину элемента. Для этого мы напишем:

```
.username {  
    white-space: nowrap; /* Запрещаем перенос строк */  
    overflow: hidden; /* Обрезаем все, что не помещается в область */  
    text-overflow: ellipsis; /* Добавляем многоточие в конце */  
}
```



Другое дело!

## Предотвращение искажения изображения

Если изображение не соответствует размерам, которые мы указали в CSS, оно по умолчанию будет сжиматься или растягиваться. Рассмотрим пример - вот карточка еды, в которой изображение соответствует размерам области:



**Dawood Basha**

Meat balls with tomato sauce

Когда пользователь загружает изображение другого размера, оно будет искажено. Это нехорошо! Посмотрите, как растянуто изображение:



Исходное фото



Искаженное фото



Dawood Basha

Meat balls with tomato sauce

фото искажается, т.к. пытается заполнить всю область карточки целиком

Чтобы изображение не искажалось. мы используем `object-fit`:

```
.card__image { object-fit: cover; }
```

На уровне проекта я предпогитаю добавлять `object-fit` ко **всем изображениям**, чтобы избежать неожиданных результатов изображения.

```
img { object-fit: cover; }
```

Обратите внимание, что изображение заполняет собой всю область `img`, а те части изображения которые не умещаются, мы не увидим.

## Блокировка цепочки прокрутки

Вы когда-нибудь открывали модальное окно и начинали скроллить, а затем, когда вы доходите до конца и продолжаете прокручивать, содержимое под модальным окном (элемент `body`) будет прокручиваться? Это называется **цепочкой прокрутки** (`scroll chaining`).

В последние годы было несколько хаков, чтобы избежать этого, но теперь мы можем сделать это только с помощью CSS, благодаря свойству `overscroll-behavior`

На следующем рисунке вы видите поведение цепочки прокрутки по умолчанию.

## overscroll-behavior: **auto**

When we reach the modal boundary, the scrolling  
will start on the body content



Чтобы подобной ситуации не возникало, мы добавим следующие стили к любому компоненту, который нуждается в прокрутке (например, компонент чата, мобильное меню и т. д.). Преимущество этих свойств в том, что они никак себя не проявляют, если прокрутка вовсе отсутствует.

```
.modal__content {  
    overscroll-behavior-y: contain;  
    overflow-y: auto;  
}
```

## Резервное значение CSS-переменной

Переменные CSS все чаще используются в веб-дизайне. Существует метод, который мы можем применить чтобы вёрстка не поплыла, если значение переменной CSS по какой-то причине было пустым.

Это особенно полезно при передаче значения переменной CSS через JavaScript. Вот пример:

```
.message__bubble {  
    max-width: calc(100% - var(--actions-width));  
}
```

Переменная `--actions-width` используется внутри `calc()` функции, и ее значение исходит из JavaScript. Предположим, что JavaScript по какой-то причине дал сбой, что

JavaScipt. Предположим, что JavaScipt по какой-то причине дал свой, что

произойдет? Будет `max-width` ошибочно назначено `none`.

Мы можем заранее избежать этого добавив резервное значение в конструкцию `var()`.

```
.message__bubble {  
    max-width: calc(100% - var(--actions-width, 70px));  
}
```

Таким образом, если переменная не определена, будет использован резервный вариант - `70px`. Этот подход нужно использовать только когда существует вероятность, что переменная может выйти из строя (например, исходящая из Javascript). В противном случае он не нужен.

## Использование фиксированной ширины или высоты

Одной из распространенных вещей, которые ломают макет, является использование фиксированной ширины или высоты с элементом, который имеет содержимое разной длины.

## Фиксированная высота

Я часто вижу элементы с фиксированной высотой и содержимым, превышающим эту высоту, что приводит к нарушению макета. Не знаете, как это выглядит? Вот.

```
.block {  
    height: 350px;  
}
```

## Defensive CSS

Логотипы автомобилей и автобусов

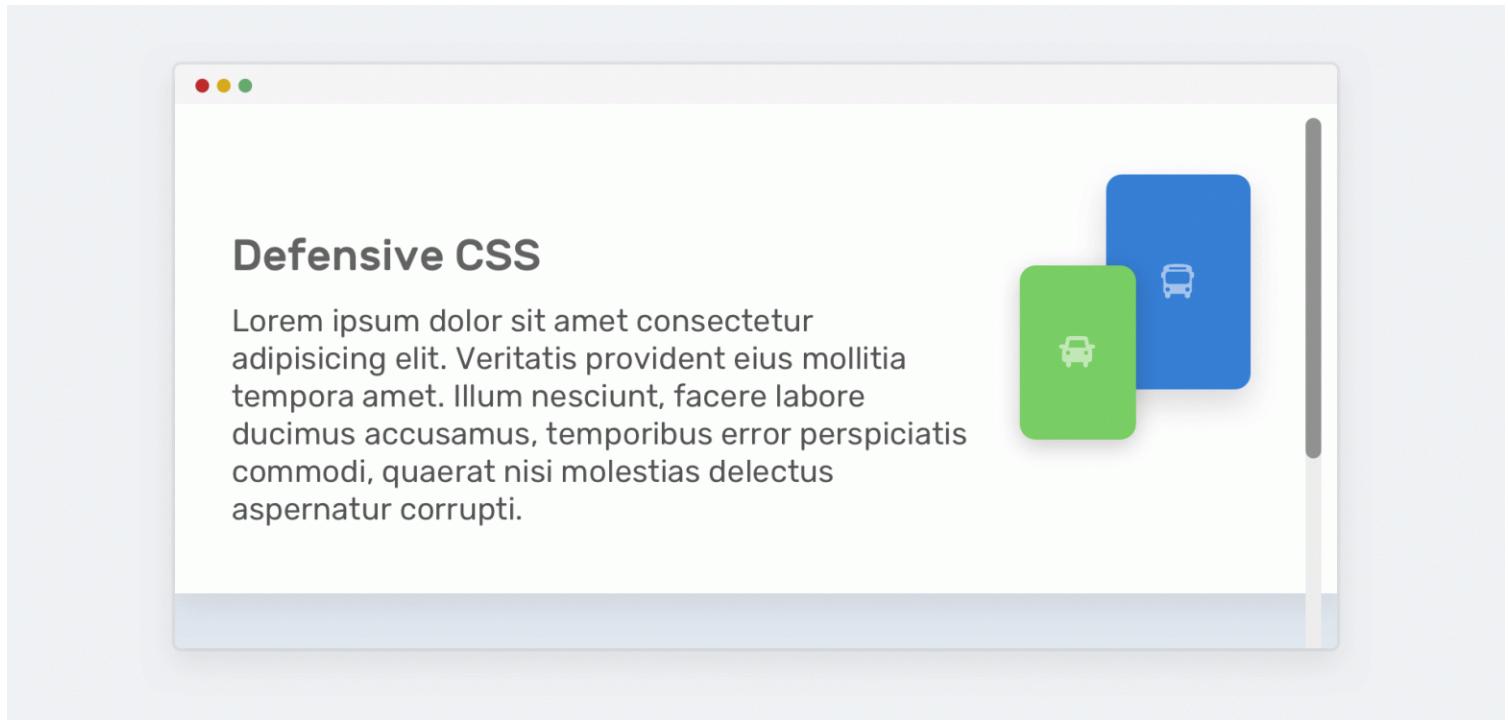
Defensive CSS

Lorem ipsum dolor sit amet consectetur  
adipisicing elit. Veritatis provident eius mollitia  
tempora amet. Illum nesciunt, facere labore  
ducimus accusamus, temporibus error perspiciatis  
commodi, quaerat nisi molestias delectus  
aspernatur corrupti.



Чтобы избежать утечки контента из блока, нам нужно использовать `min-height` вместо `height`:

```
.block {  
  min-height: 350px;  
}
```



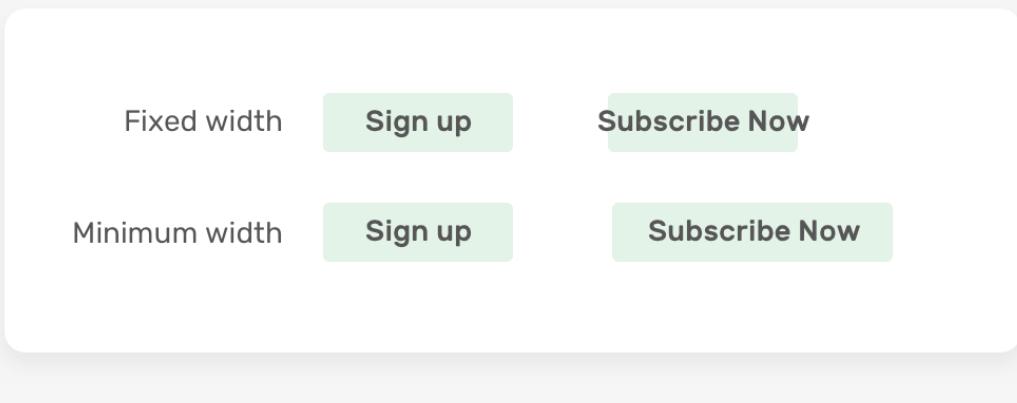
Таким образом, если содержимое станет больше, макет не сломается - контейнер станет больше по высоте

## Фиксированная ширина

Вы когда-нибудь видели кнопку, контент которой расположен слишком близко к левому и правому краям? Это связано с использованием фиксированной ширины.

```
.button {  
  width: 100px;  
}
```

Если текст кнопки длиннее 100px, он будет близко к краям. Если текст слишком длинный, он будет просачиваться из кнопки. Это нехорошо!



Чтобы исправить это, мы можем просто заменить `width` на `min-width`.

```
.button {  
  min-width: 100px;  
}
```

## Отключение повторения фона

Зачастую при использовании большого изображения в качестве фона мы забываем учитывать случай, когда дизайн просматривается на большом экране. Этот фон будет **повторяться** по умолчанию.

В основном это не будет видно на экране ноутбука, но его можно четко увидеть на больших экранах.



Маленький экран



Большой экран



Изображение повторяется 😞

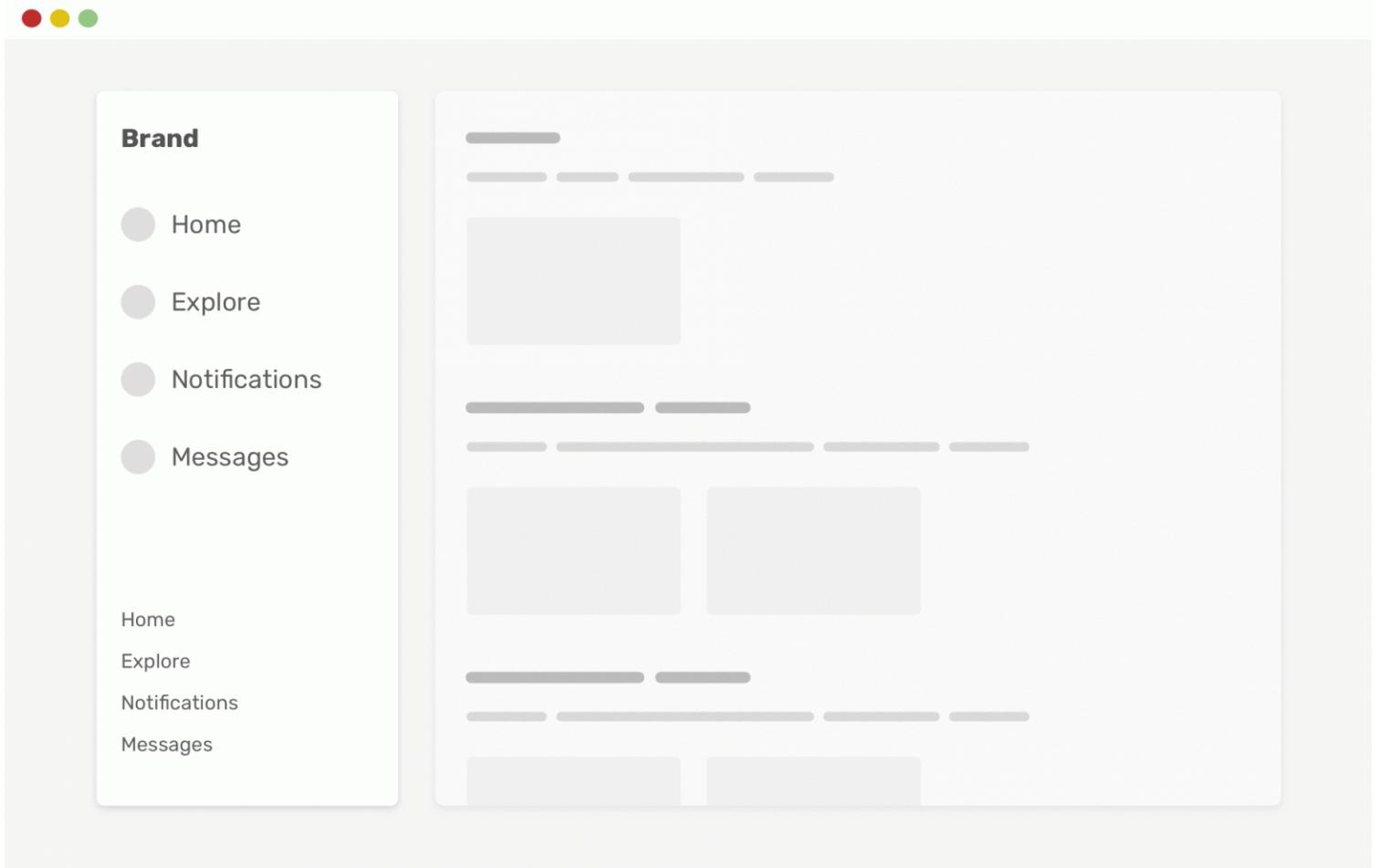
Чтобы заранее избежать такого поведения, обязательно нужно написать, что фон повторяться не должен!

```
.image {  
  background-image: url('..');  
  background-repeat: no-repeat; /* запрет повторения изображения */  
}
```

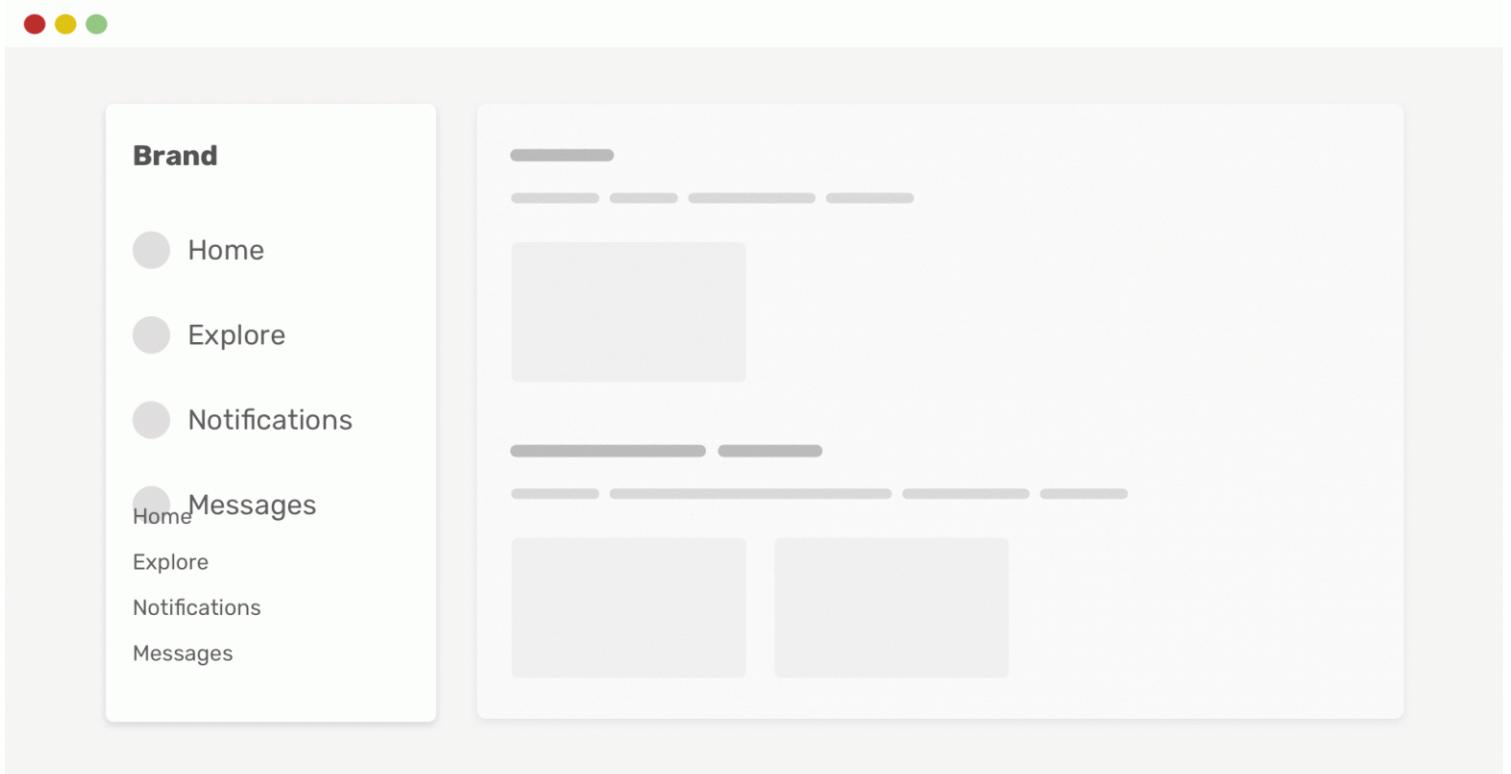
## Вертикальные @media-запросы

Конечно проще сверстать компонент и протестировать его, только меняя ширину браузера. Однако тестирование по высоте браузера может выявить некоторые интересные проблемы.

Вот пример, который я видел несколько раз. У нас есть боковая секция с основным и второстепенным меню. Второстепенное меню должно располагаться в самом низу боковой секции. Разработчик добавил `position: sticky` к второстепенному меню, чтобы оно могло прилипать к низу. На первый взгляд всё хорошо:



Проблема в том, что если высота браузера будет меньше, все сломается. Обратите внимание, как два меню наползают друг на друга:



Используя вертикальные медиа-запросы CSS, мы можем избежать этой проблемы.

```
@media (min-height: 600px) {  
    .secondary_menu {  
        position: sticky;  
        bottom: 0;  
    }  
}
```

Таким образом, второстепенное меню будет привязано к нижней части **только в том случае, если высота области просмотра больше или равна 600px**. Гораздо лучше, правда?

Вероятно, есть лучшие способы реализации такого поведения (например, использование `margin-auto`), но в этом примере я сосредоточился на вертикальном запросе.

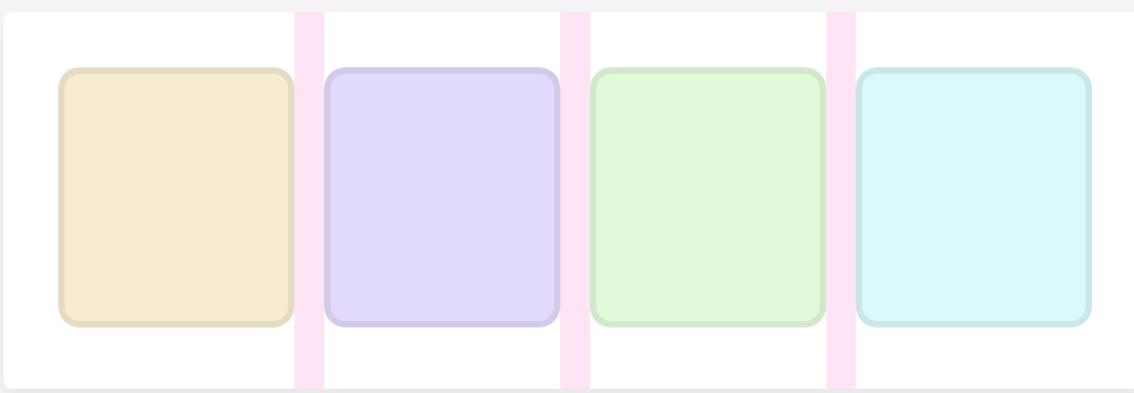
## Использование `justify-content: space-between`

В flex-контейнере вы можете использовать `justify-content` для разделения дочерних элементов друг от друга. При определенном количестве дочерних элементов макет будет выглядеть normally. Однако, когда они увеличиваются или уменьшаются, макет будет выглядеть странно.

Рассмотрим следующий пример:



`justify-content: space-between`



У нас есть гибкий контейнер с четырьмя элементами. Расстояние между каждым элементом не является следствием использования `gap` или `margin`, оно существует, потому что в контейнере есть `justify-content: space-between`.

```
.wrapper {
```

```
    display: flex;
```

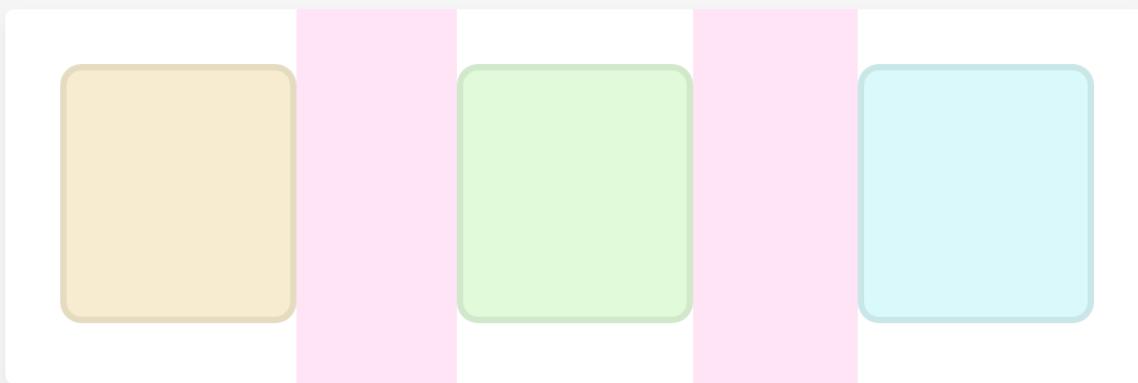
```
    justify-content: space-between;
```

```
display: flex;  
flex-wrap: wrap;  
justify-content: space-between;  
}
```

Когда количество элементов меньше четырех, вот что произойдет:



justify-content: space-between



Это нехорошо. У данной проблемы есть несколько разных решений

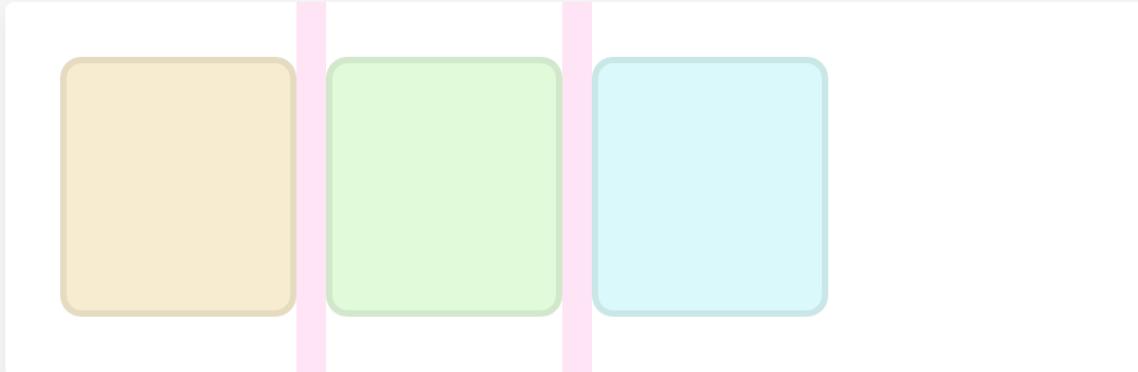
- Margin
- Flexbox gap (используйте с осторожностью)
- Padding (может применяться к родительскому элементу каждого дочернего элемента)
- Добавление пустых элементов в качестве разделителя

Для простоты я буду использовать gap :

```
.wrapper {  
  display: flex;  
  flex-wrap: wrap;  
  gap: 1rem;  
}
```



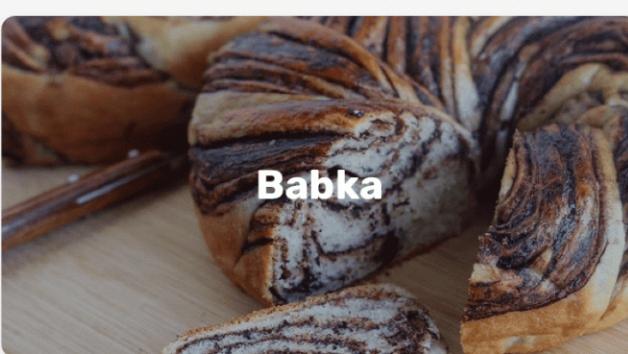
gap: 1rem



## Текст поверх изображений

При использовании текста поверх изображения важно учитывать случай, когда изображение не загружается. Как будет выглядеть текст?

Вот пример:



Текст выглядит читаемым, но когда изображение не загружается, белый текст теряется на фоне белой страницы:

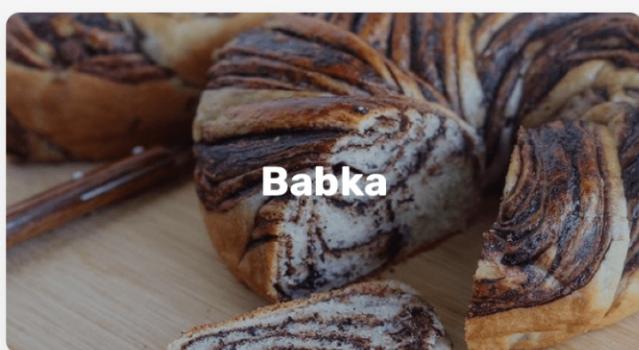


Мы легко исправим это, добавив фоновый цвет к `<img>` элементу. Этот фон будет виден только в том случае, если изображение не загрузится. Разве это не круто?

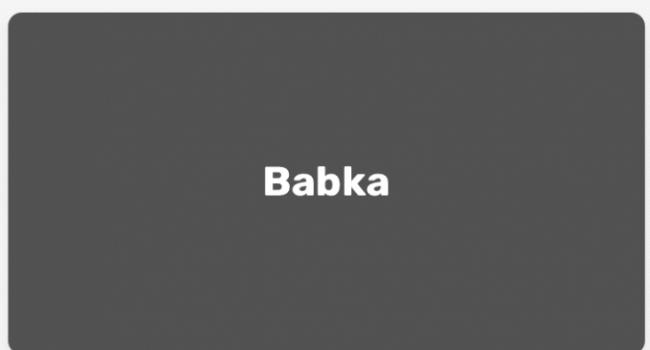
```
.card__img {  
background-color: grey;  
}
```



Изображение загрузилось



Изображение не загрузилось



## Будьте осторожны с фиксированными значениями в CSS-Grid

Скажем, у нас есть сетка, которая содержит боковую секцию и основную. CSS выглядит так:

```
.wrapper {
```

```
... wrapper {  
    display: grid;  
  
    grid-template-columns: 250px 1fr;  
    gap: 1rem;  
}
```

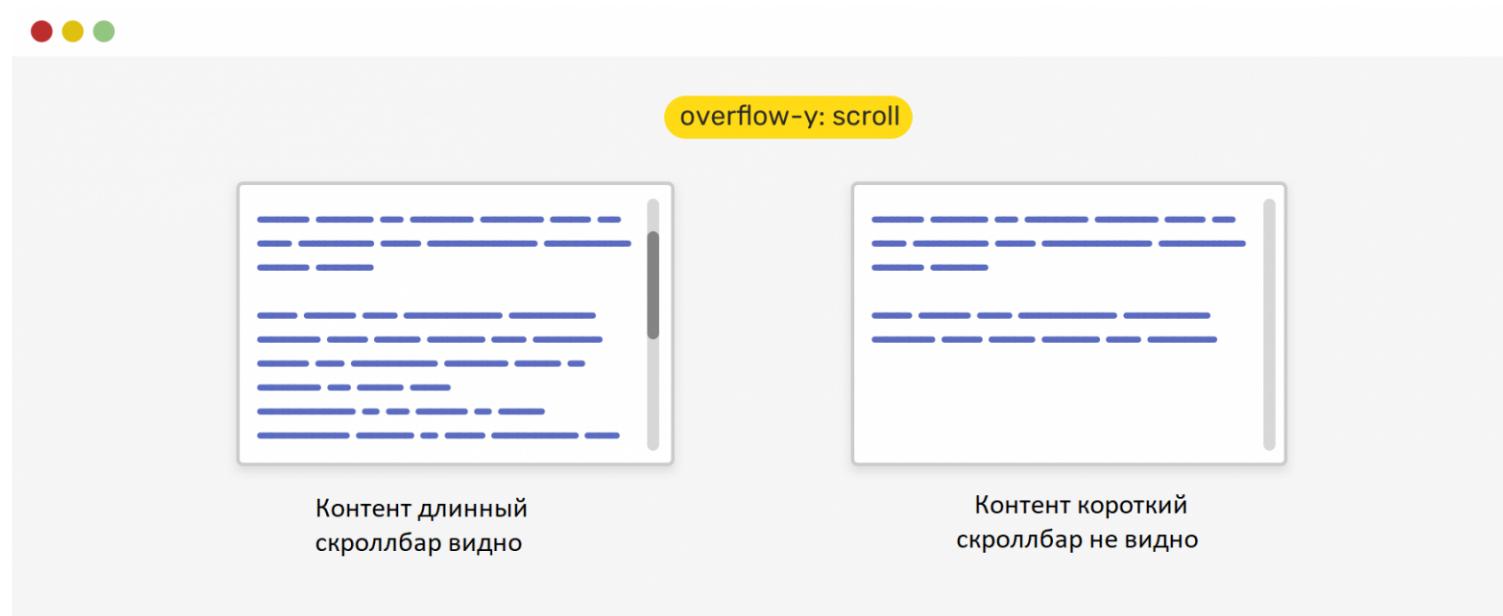
Вёрстка поплынет при небольших размерах окна просмотра из-за нехватки места. Чтобы избежать такой проблемы, всегда используйте медиа-запрос при использовании сетки CSS, как на примере ниже:

```
@media (min-width: 600px) {  
    .wrapper {  
        display: grid;  
        grid-template-columns: 250px 1fr;  
        gap: 1rem;  
    }  
}
```

## Показываем полосу прокрутки только когда это необходимо

К счастью, мы можем управлять отображением полосы прокрутки или ее отсутствием не только в случае наличия длинного контента. Настоятельно рекомендуется использовать `auto` в качестве значения для `overflow`.

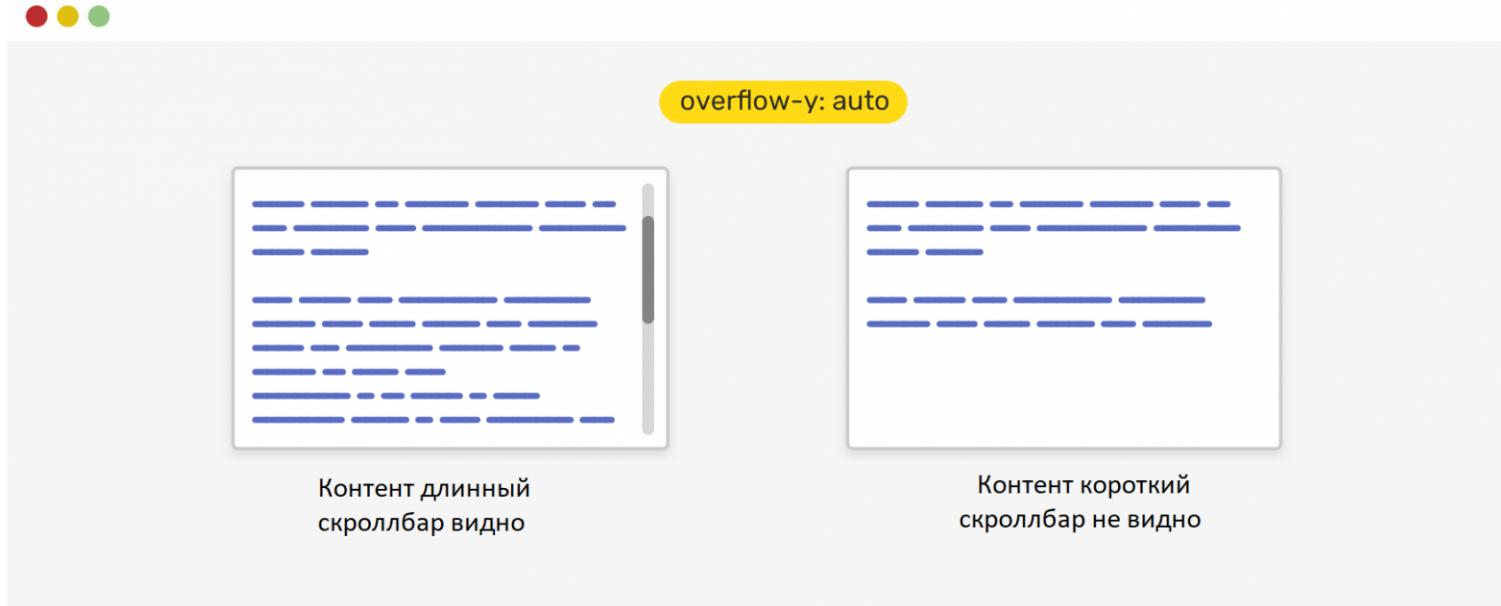
Рассмотрим следующий пример:



Обратите внимание, что даже если содержимое короткое, полоса прокрутки видна. Это не хорошо для пользовательского интерфейса. Как дизайнер, я просто сбиваюсь с толку, видя полосу прокрутки, когда она не нужна.

При `overflow-y: auto` полоса прокрутки будет видна только в том случае, если содержимое длинное. Иначе её там не будет:

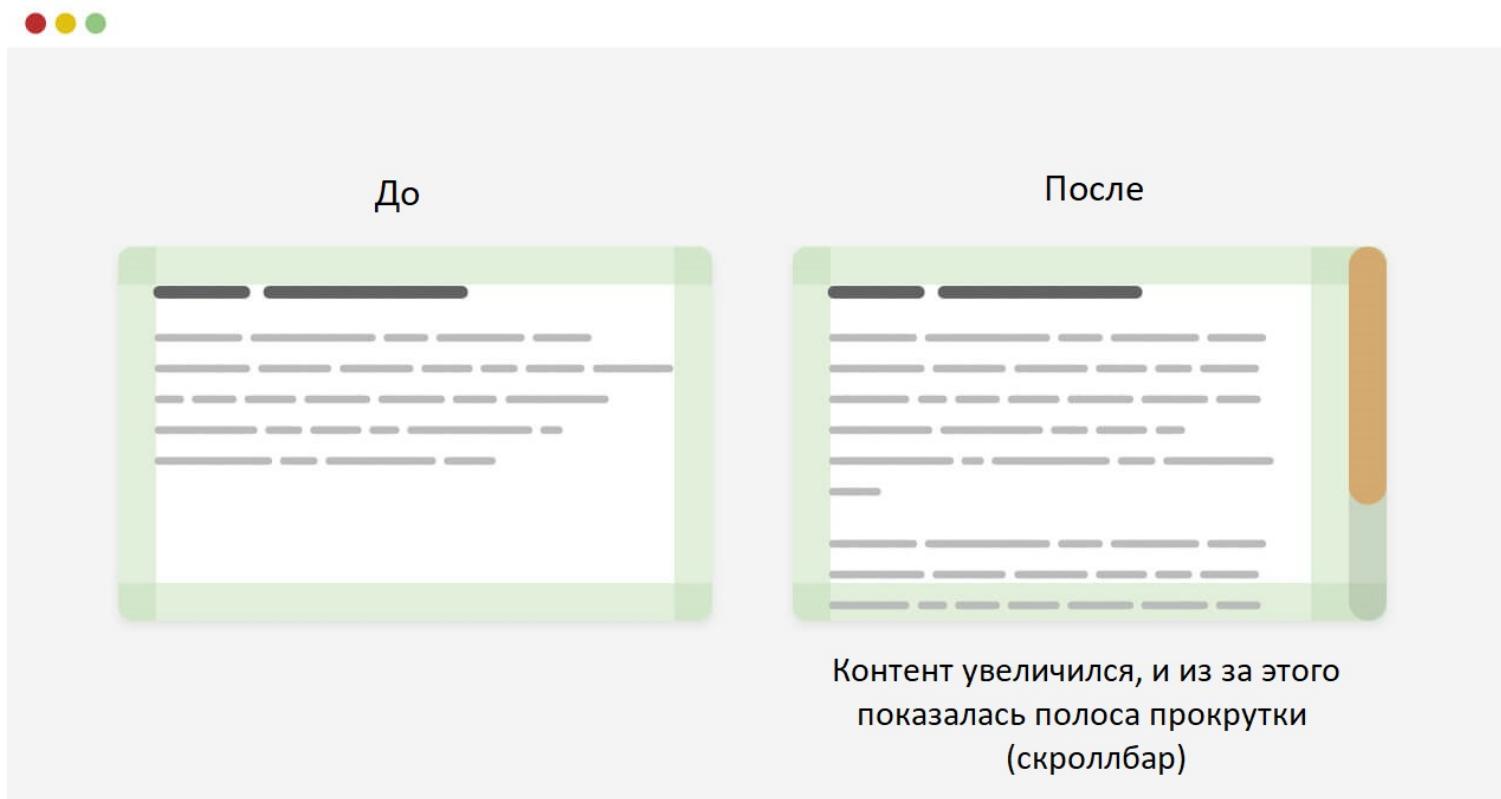
```
.element {  
    overflow-y: auto;  
}
```



## Ширина контента с полосой прокрутки

Еще одна вещь, связанная с прокруткой - её ширина. Взяв предыдущий пример, когда содержимое становится длиннее, добавление полосы прокрутки вызовет сдвиг макета. Причина смены макета заключается в том, чтобы зарезервировать место для полосы прокрутки.

Рассмотрим следующий пример:



Обратите внимание, как сдвинулось в ширину содержимое, в результате добавления полосы прокрутки. Мы можем зафиксировать ширину контента, чтобы скроллбар её не менял. Для этого напишем стили:

```
.element {  
    scrollbar-gutter: stable;  
}
```



Короткий контент



зарезервировано  
место для скроллбара

Длинный контент



при появлении скроллбара  
ширина контента не изменилась!

## Минимальный размер контента в CSS-Flexbox

Если во flex-контейнере есть текстовый элемент или изображение, длина которого превышает длину самого элемента, браузер не будет их уменьшать. Это поведение по умолчанию для flexbox.

Рассмотрим следующий пример:

```
.card {  
    display: flex;  
}
```

Если в заголовке очень длинное слово, оно не будет переноситься на новую строку.



Даже если мы используем `overflow-wrap: break-word`, это не сработает.

Чтобы изменить это поведение, нам нужно установить `min-width` дочернего блока на `0`, из-за того, что значение по умолчанию `min-width` равно `auto`.

```
.card__title {  
    overflow-wrap: break-word;  
    min-width: 0;  
}
```

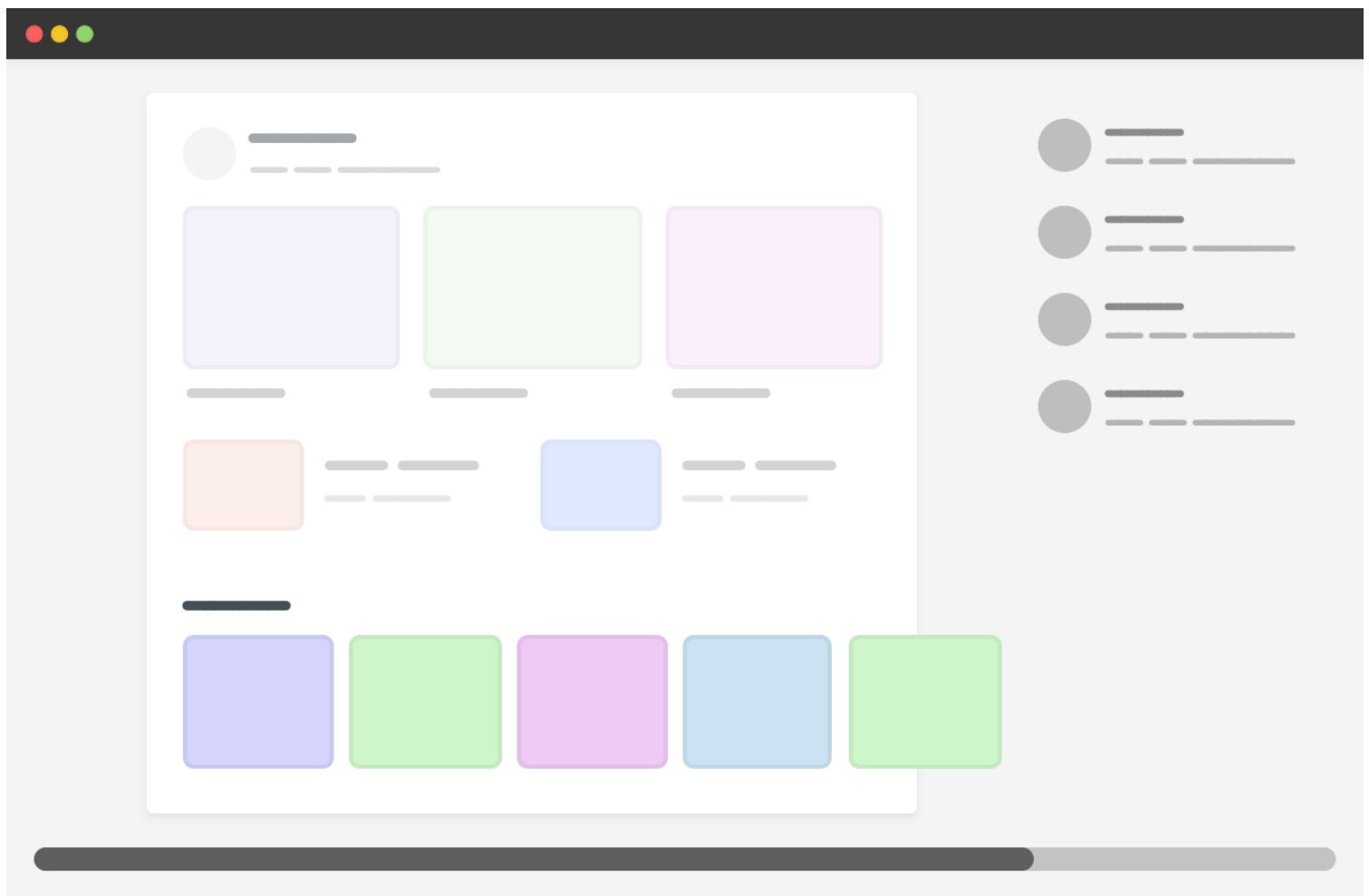
То же самое относится и к flex-контейнеру, но вместо `min-width: 0` мы будем использовать `min-height: 0`. Результат:



## Минимальный размер содержимого в CSS-Grid

Подобно flexbox, CSS-сетка имеет минимальный размер содержимого по умолчанию для своих

дочерних элементов, который равен auto . Это означает, что если есть элемент, который больше, чем элемент сетки, сетка переполнится.



В приведенном выше примере у нас есть карусель в основном разделе. Для контекста, вот HTML и CSS примера:

```
<div class="wrapper">  
    <main>  
        <section class="carousel"></section>  
    </main>  
  
    <aside></aside>  
  
</div>
```

```
@media (min-width: 1020px) {  
    .wrapper {  
        display: grid;  
        grid-template-columns: 1fr 248px;  
        grid-gap: 40px;  
    }  
}
```

```
}
```

```
.carousel {  
    display: flex;  
    overflow-x: auto;  
}
```

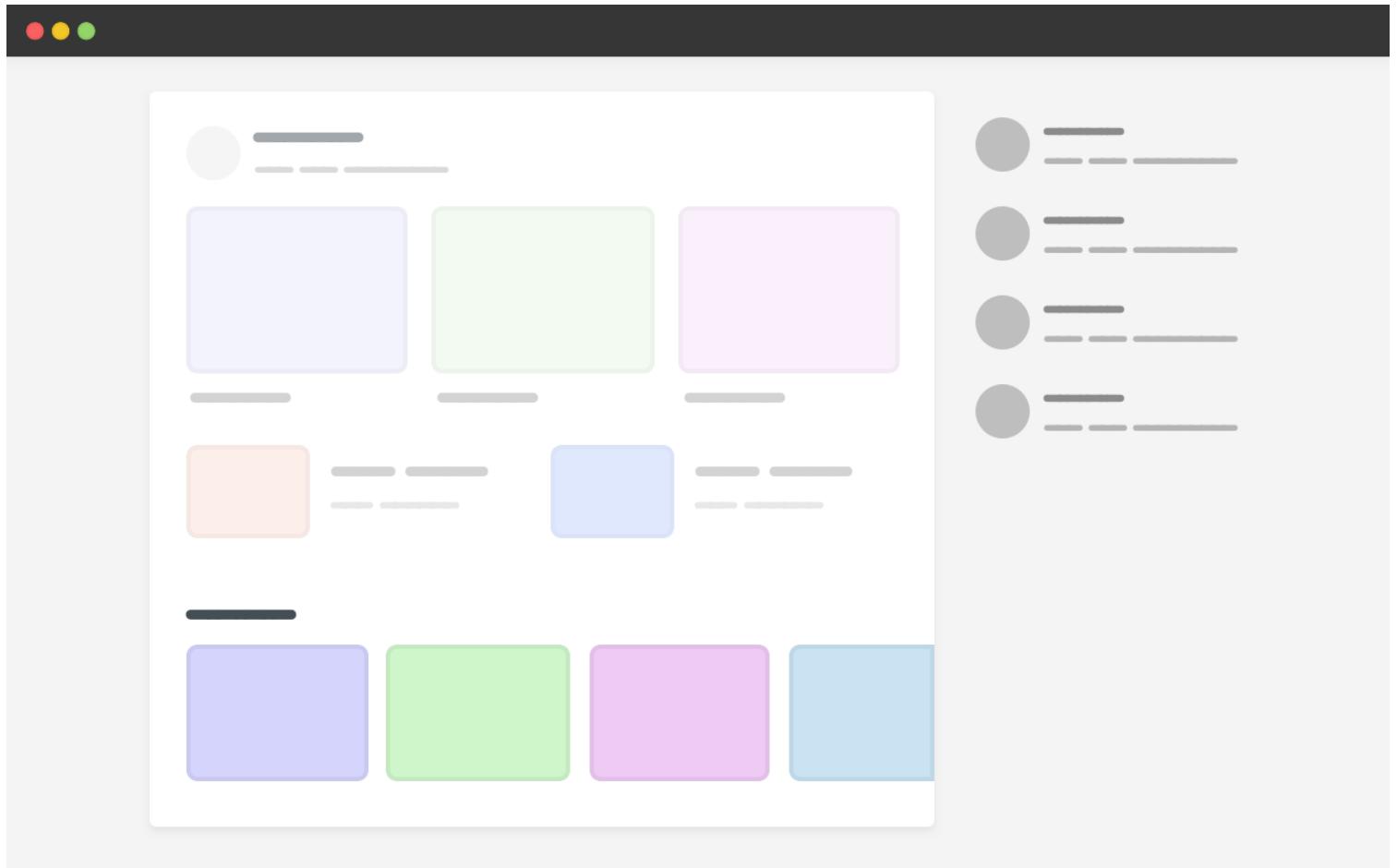
Поскольку карусель представляет собой flex-контейнер, который не переносит собственное содержимое на другую строку, её ширина больше, чем секция в которой она находится, в результате этого присутствует горизонтальная прокрутка.

Чтобы исправить это, у нас есть три разных решения:

- Использование `minmax()`
- Применение `min-width` к элементу сетки
- Добавление `overflow: hidden` элемента в сетку

В качестве "безопасного" CSS я бы выбрал вариант который использует функцию `minmax()`

```
@media (min-width: 1020px) {  
    .wrapper {  
        display: grid;  
        grid-template-columns: minmax(0, 1fr) 248px;  
        grid-gap: 40px;  
    }  
}
```



## Auto Fit против Auto Fill

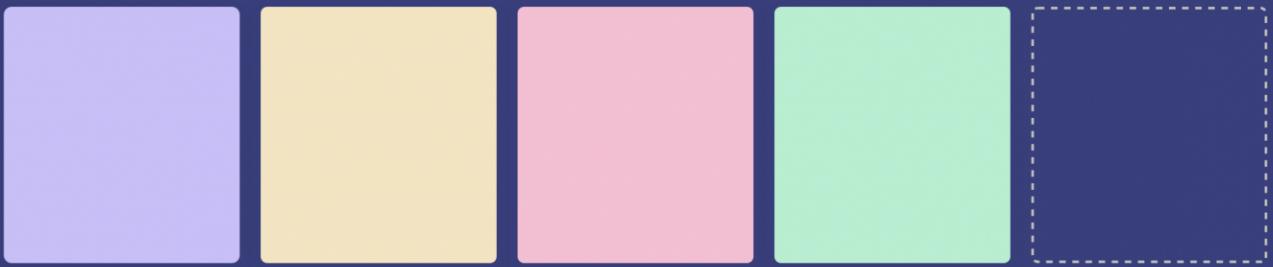
При использовании функции CSS-Grid `minmax()` важно выбрать между использованием параметров `auto-fit` или `auto-fill`. При неправильном использовании это может привести к неожиданным результатам.

При использовании функции `minmax()` параметр `auto-fit` расширяет элементы сетки, чтобы заполнить доступное пространство. В свою очередь `auto-fill` сохранит доступное пространство зарезервированным, без изменения ширины элементов сетки. Наглядный пример:

auto-fill

```
repeat(auto-fill, minmax(200px, 1fr));
```

сохранит доступное пространство зарезервированным, без изменения ширины элементов сетки



auto-fit

```
repeat(auto-fit, minmax(200px, 1fr));
```

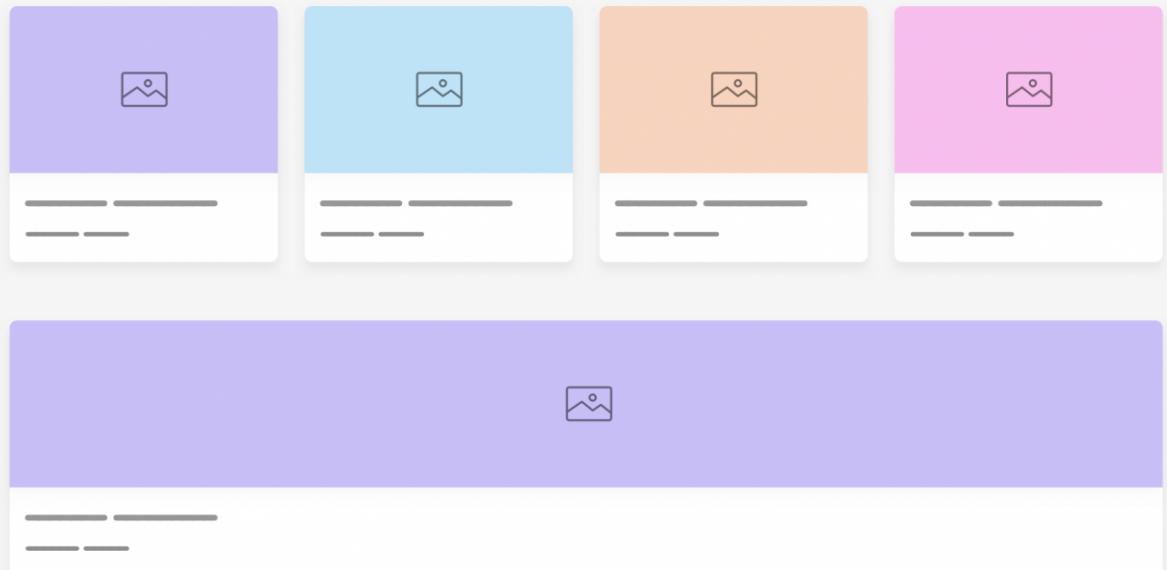
расширяет элементы сетки, чтобы заполнить доступное пространство



При этом, использование `auto-fit` может привести к тому, что элементы сетки будут слишком широкими, особенно если они меньше ожидаемого. Рассмотрим следующий пример.

```
.wrapper {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
  grid-gap: 1rem;  
}
```

Если используется только один элемент сетки `auto-fit`, он будет расширяться, чтобы заполнить ширину контейнера:



В большинстве случаев такое поведение не требуется, поэтому лучше использовать `auto-fill`

```
.wrapper {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));  
  grid-gap: 1rem;  
}
```



Не забудьте установить `max-width: 100%` для **всех** изображений, чтобы они не вылезали за пределы блоков, в которых находятся. Это можно добавить к предыдущим глобальным стилям для `img`, вот что получится:

```
img {  
  max-width: 100%;  
  object-fit: cover;  
}
```

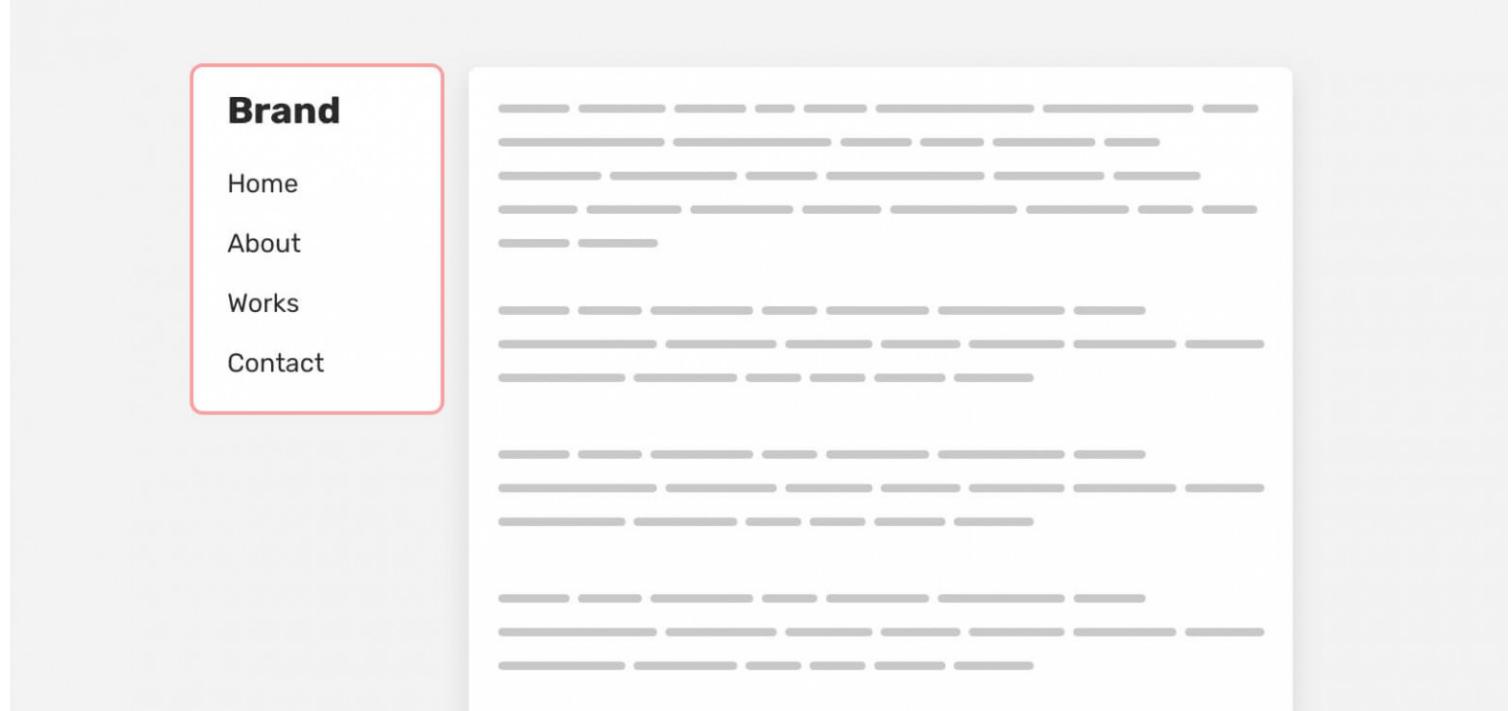
## Использование position: sticky в CSS-Grid

Вы когда-нибудь пробовали использовать `position: sticky` с дочерним элементом grid-контейнера? По умолчанию, элементы grid **растягиваются**. В результате боковой элемент в приведенном ниже примере равен высоте основной секции.



Чтобы заставить его работать должным образом, вам нужно сбросить свойство `align-self`

```
aside {  
  align-self: start;  
  position: sticky;  
  top: 1rem;  
}
```



## Группировка селекторов для разных браузеров

Не рекомендуется группировать селекторы, предназначенные для работы с разными браузерами. Например, для стилизации заполнителя ввода требуется несколько селекторов для каждого браузера. Если мы сгруппируем селекторы, согласно w3c **все правило будет недействительным**.

```
/* Не делайте так, пожалуйста! */
input::-webkit-input-placeholder,
input:-moz-placeholder {
    color: #222;
}
```

Вместо этого напишите так:

```
input::-webkit-input-placeholder {
    color: #222;
}

input:-moz-placeholder {
    color: #222;
}
```

Это список безопасных техник CSS, которые я постоянно использую в своих проектах.

**Спасибо Ахмаду Шадиду за качественный контент!**

(P.S.: Во многих местах я перевёл своими словами, более доступно для русскоговорящего читателя.)

От переводчика: подытоживая текст статьи, и многолетний опыт вёрстки, хочу дать совет читателям верстальщикам: когда верстаете, всегда думайте о том, какие ситуации могут возникнуть при использовании вашей вёрстки. Что будет, если текста будет сильно больше или сильно меньше. Как элементы себя поведут на устройствах с разной шириной и высотой экранов (кстати для тестирования последнего рекомендую бесплатную программу Responsively). Страйтесь избегать дублирования кода - создавайте общие css-классы для часто используемых стилей или наборов стилей. Используйте препроцессоры (SCSS например), если ещё не использовали их, поверьте, это очень удобно. И самое главное - наслаждайтесь процессом!

**Теги:** html, css, adaptive, flexbox, grid, flex, вёрстка, верстка, стили

**Хабы:** CSS, HTML

## Редакторский дайджест

Присыпаем лучшие статьи раз в месяц

Электропочта



17

0

Карма Рейтинг

**Андрей Рик @Rikkster**

full-stack developer

Реклама



▼ onebusiness.ru РЕКЛАМА

Создавайте сайт бесплатно  
и продвигайте бизнес онлайн



⌚ webinar-pro.ru РЕКЛАМА · 16+

Научись зарабатывать на Авито!  
Бесплатный вебинар

**Комментарии 12**



○ nin-jin  
30.01.2022 в 00:44

Важно, чтобы элементы выглядели однотипно. Для этого мы должны запретить перенос слов, и обрезать часть слов, если они выходят за ширину элемента.

Кому важно? Пользователю важно, чтобы он мог прочитать текст полностью, а не два с половиной слова.

Чтобы изображение неискажалось, мы используем object-fit: cover

В результате, загружает пользователь вертикальное фото, а у сверху треть обрезана и снизу третья. А через замочную скважину не разобрать что же изображено на картинке. Зато дизайнер счастлив.

Мы можем заранее избежать этого добавив резервное значение в конструкцию var()

Потом это резервное значение разъедется с актуальным дизайном и разработчик будет счастлив прошерстить все стили в проекте, чтобы поменять резервные значения. Просто пропишите эти значения для :root и всё.

При overflow-y: auto полоса прокрутки будет видна только в том случае, если содержимое длинное. Иначе её там не будет

А при динамическом изменении высоты контента или контейнера будет прыгать туда-сюда и ширина контента. Лучше тогда уж overflow: overlay .

+10 Ответить

Fayon  
31.01.2022 в 00:22

Лучше тогда уж overflow: overlay

<https://caniuse.com/css-overflow-overlay> - несомненно лучше, ага.

+2 Ответить

Apovlitos  
04.02.2022 в 17:46

Кому важно? Пользователю важно, чтобы он мог прочитать текст полностью, а не два с половиной слова.

Да, но это правило работает **только на больших текстах**. Если это в мессенджере, то тут стоит обрезать имя пользователя. Да даже в телефоне контакты обрезаются, если они слишком длинные.

А при динамическом изменении высоты контента или контейнера будет прыгать туда-сюда и ширина контента. Лучше тогда уж overflow: overlay

Подписываюсь под комментатором выше, поддержка этого свойства, на всех платформах, будет ещё нескоро

0 Ответить

nin-jin  
04.02.2022 в 18:29

Ага, в результате двух пользователей с одним префиксом в имени не можешь отличить, пока не зайдёшь к каждому в профиль. Зато дизайнер счастлив.

0 Ответить

art\_code  
30.01.2022 в 00:47

Полезно. Очень часто замечаю, что разработчики даже не задумываются над тем, что будет, если просто

поменять количество текста в блоке.

+2 Ответить



**kawena54**  
30.01.2022 в 15:10

"img { object-fit: cover; } "

это невероятно плохо работает когда каринки заметно вытянуты по вертикали и ужи чем ожидалось

"Не забудьте установить max-width: 100% для **всех** изображений, "

а по умолчанию как это работает ?

0 Ответить



**Rikkster**  
30.01.2022 в 15:11

по умолчанию img-изображение может быть шире чем ширина контейнера в котором оно находится

0 Ответить



**Fayon**  
31.01.2022 в 00:23

а по умолчанию как это работает ?

По умолчанию картинка грузится в актуальном размере вне зависимости от размера контейнера.

0 Ответить



**jointimer**  
31.01.2022 в 11:48

Спасибо вам большое, очень помогло.

0 Ответить



**wadowad**  
01.02.2022 в 05:39

Красиво конечно, когда изображение с помощью object-fit: cover вписывается в определённую область. Но в статье столько сказано про универсальность решений, а тут провал. Дело в том, что всегда будут картинки разной ориентации. И в одном случае изображения будут обрезаться по бокам, а в другом сверху и снизу. В первую очередь нужно исходить от того, какой контент предполагается в данном месте. Например, если это карточки с фотографиями людей, то можно попробовать object-position: top (или процентное значение), чтобы головы не обрезались на вертикальных фотках. Если же на сайте предполагается, что будут и вертикальные и горизонтальные картинки, тогда лучше придумать решение с object-fit: contain и каким-либо фоном карточки.

+1 Ответить



**rexen**  
07.03.2022 в 08:56

Кстати, у Вордпресса есть какая-то галерея-плагин, где при апложде картинок можно указать

координаты "фокуса" на изображении. Тогда система при обрезке картинок до нужных "плиток" учитывает, какую часть фотки нужно обязательно оставить на виду.

0 Ответить



NikolasPushkin

04.02.2022 в 14:11

Спасибо, когда заходил по ссылке думал что будет очередная статья о выравнивании дива по вертикали, но меня очень порадовали свойства overscroll-behavior и scrollbar-gutter, единственное замечание - я бы добавил в статью ссылки на поддержку этих свойств( к сожалению пока не везде это работает хорошо). Верстаю давно и ранее такие моменты обходил с помощью хаков, теперь будет для нормальных браузеров стандартные решения.

+1 Ответить



Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

## Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ



AlekDikarev сегодня в 11:02

**Мы разработали 44 устройства за 6 лет, продаем их по всему миру, только этого мало**

0 +147

13K

35

39 +39



ermakovd вчера в 22:43

**Система увлажнения воздуха для дома или квартиры**

Tutorial

0 +67

10K

111

57 +57



YuriPanchul сегодня в 11:49

**Что делать, когда выпускник топ-10 мирового вуза не может спроектировать блок сложения A+B**

0 +37

13K

32

25 +25



trjful вчера в 23:50

## У электронной промышленности Китая проблемы с архитектурой ARM и NAND-чипами. Какие у Поднебесной шансы



+34

7.1K

12

15 +15

InBioReactor вчера в 20:05

### Мишель Сифр. Замурованный



+32

4.2K

22

42 +42

### Отобразить любой контент как bottom sheet: турориал на грани света и тьмы

Турбо

#### МИНУТОЧКУ ВНИМАНИЯ

Разместить



Мегапост

Заглядываем в будущее Data Mining с хабрвчанами



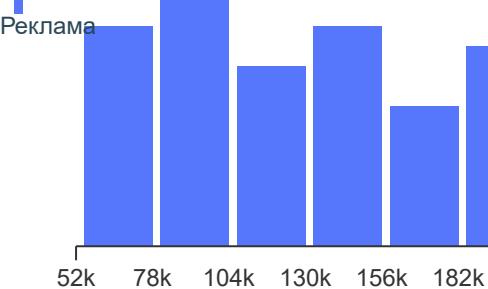
Турбо ↗

Как небольшой компании расцвести на Хабре за полгода

#### СРЕДНЯЯ ЗАРПЛАТА В ИТ

**160 151 ₽/мес.**

— средняя зарплата во всех IT-специализациях по данным из 9 762 анкет, за 2-ое пол. 2022 года. Проверьте «в рынке» ли ваша зарплата или нет!



Проверить свою зарплату

РЕКЛАМА · 16+ Я

**Postman**

Я Практикум

Узнать больше

[practicum.yandex.ru](https://practicum.yandex.ru)

## Курс: Инженер по тестированию. Начните учиться бесплатно

Онлайн-курсы от Яндекса

### ЧИТАЮТ СЕЙЧАС

Физик создал математическую модель путешествий во времени без парадоксов

6.5K 80 **+80**

Что делать, когда выпускник топ-10 мирового вуза не может спроектировать блок сложения A+B

13K 25 **+25**

Мы разработали 44 устройства за 6 лет, продаем их по всему миру, только этого мало

13K 39 **+39**

SimpleX – первый мессенджер без идентификаторов пользователей

3.6K 12 **+12**

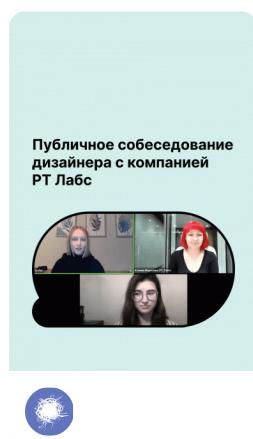
## Как получить доступ к chatGPT в России

82K 71 +71

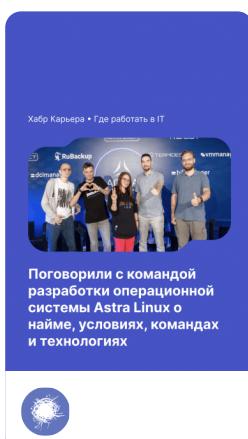
Поём оды экспертизе и хардкору в дуэте с вами на новом проекте Хабра

Турбо

## ИСТОРИИ



Публичное  
собеседование:  
дизайнер



Хабр Карьера • Где работать в IT



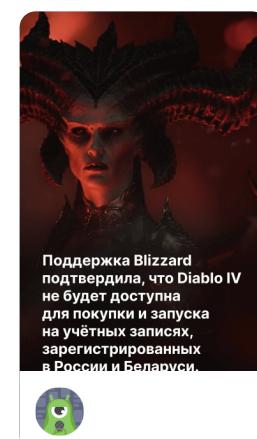
Поговорили с командой разработки операционной системы Astra Linux о найме, условиях, командах и технологиях



Европейская организация по ядерным исследованиям (ЦЕРН) и Национальная ускорительная лаборатория имени Энрико Ферми (Фермилаб) заявили о переходе на AlmaLinux.



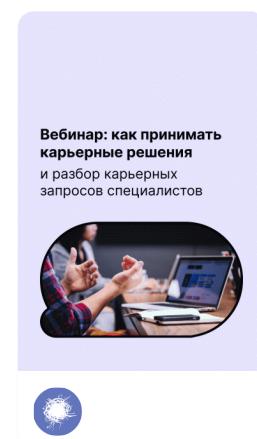
ЦЕРН и Фермилаб  
перешли на  
AlmaLinux



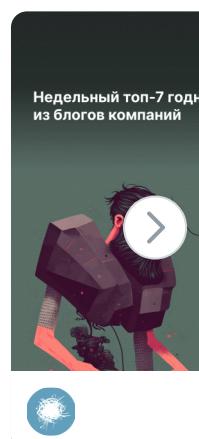
Поддержка Blizzard подтвердила, что Diablo IV не будет доступна для покупки и запуска на учётных записях, зарегистрированных в России и Беларусь.



Diablo IV не  
появится в РФ



Вебинар: как  
принимать  
карьерные  
решения



Недельный топ-7 годноты от  
компаний

Реклама

GETINTENT

Ваш аккаунт

Разделы

Информация

Услуги

Войти

Публикации

Устройство сайта

Корпоративный блог

Регистрация

Новости

Для авторов

Медийная реклама

Хабы

Для компаний

Нативные проекты

Компании

Документы

Образовательные

Авторы

Соглашение

программы

Песочница

Конфиденциальность

Стартапам

Мегапроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию

© 2006–2022, Habr