**Свойство @container**

@container — это новая директива CSS, которая предназначена для создания контейнеров, которые автоматически подстраиваются под ширину своего содержимого. Она представляет собой удобный инструмент для создания адаптивных и отзывчивых макетов. Контейнеры, созданные с помощью @container, могут быть использованы для размещения текста, изображений и других элементов веб-страницы.

Свойство container-type

Это свойство назначает элементу роль контейнера. Значение этого свойства определяет, относительно изменения какого свойства контейнера будет меняться внутренний элемент.

.box-container

container-type: inline-size;

}

Свойство container-type может принимать следующие значения:

inline-size: создает контейнер, запросы к которому будут учитывать его длину;

size: создает контейнер, запросы к которому будут учитывать его длину и высоту;

normal: значение по-умолчанию, элемент не будет контейнером.

Пример использования этого свойства в связке с адаптацией внутреннего элемента:

.box-container

container-type: inline-size;

}

@container (max-width: 768px) {

.box-item {

display: flex;

font-size: 16px;

}

}

Данная запись говорит о том, что элементу .box-container с помощью свойства container-type мы указали быть контейнером для использования его в контейнерных запросах дочерних элементов. В качестве значения свойства указали inline-size - значит в контейнер запросе будем опираться исключительно на его ширину. Ниже написан уже сам контейнерный запрос для дочернего элемента .box-item. Браузером эти свойства считываются так: как только ширина элемента .box-container будет <= 768px, применит для элемента .box-item внутри него свойства display:flex; и font-size: 16px;.

Свойство container-name

Это свойство указывает имя контейнеру. Используется для случаев, если внутренний элемент адаптируется относительно нескольких контейнеров.

.box-container

container-name: box; /\* имя может быть задано любое \*/

}

@container box (min-width: 768px) {

.box-item {

display: flex;

font-size: 24px;

}

}

Свойство container

Это свойство является сокращенным объединяющим вариантом для свойств container-name и container-type, которые должны быть указаны через знак /.

.box-container

container: box / size;

}

@container box (max-width: 300px) {

.box-item {

display: flex;

font-size: 14px;

}

}

**Какие проблемы решают CSS Container Queries**

Контейнерные запросы помогают найти уникальные решения для крупномасштабных проектов, в которых повторно используются компоненты, содержащиеся в разных частях веб-страниц. Поскольку запросы позволяют компонентам изменять свой внешний вид в соответствии с контекстом, в который они помещены, компоненты становятся многоразовыми и гибкими.

Примеры использования @container:

Пример 1: Создание адаптивного контейнера с новыми свойствами container-type и width

@container {

container-type: inline-size;

width: 100%;

max-inline-size: 1200px;

margin: 0 auto;

padding: 20px;

background-color: #f2f2f2;

}

В этом примере мы используем свойство container-type со значением inline-size, которое определяет размер контейнера в горизонтальном направлении в зависимости от содержимого. Мы также добавляем свойство width со значением 100%, чтобы контейнер занимал всю доступную горизонтальную ширину. Максимальная горизонтальная ширина контейнера установлена на 1200 пикселей, а отступы и фоновый цвет остаются такими же, как и в предыдущих примерах.

Пример 2: Использование вложенных контейнеров с новыми свойствами container-type и width

@container {

container-type: inline-size;

width: 100%;

max-inline-size: 900px;

margin: 0 auto;

padding: 20px;

background-color: #f2f2f2;

}

@container .nested-container {

container-type: block-size;

width: 50%;

max-block-size: 300px;

background-color: #fff;

padding: 10px;

}

В этом примере мы создаем главный контейнер с новыми свойствами container-type и width. container-type установлено на inline-size, а width на `100%, чтобы контейнер занимал всю горизонтальную ширину. Мы также создаем вложенный контейнер с новыми свойствами container-type и width .container-type установлено на block-size, а width на 50%, что означает, что вложенный контейнер займет 50% от доступного пространства в вертикальном направлении.

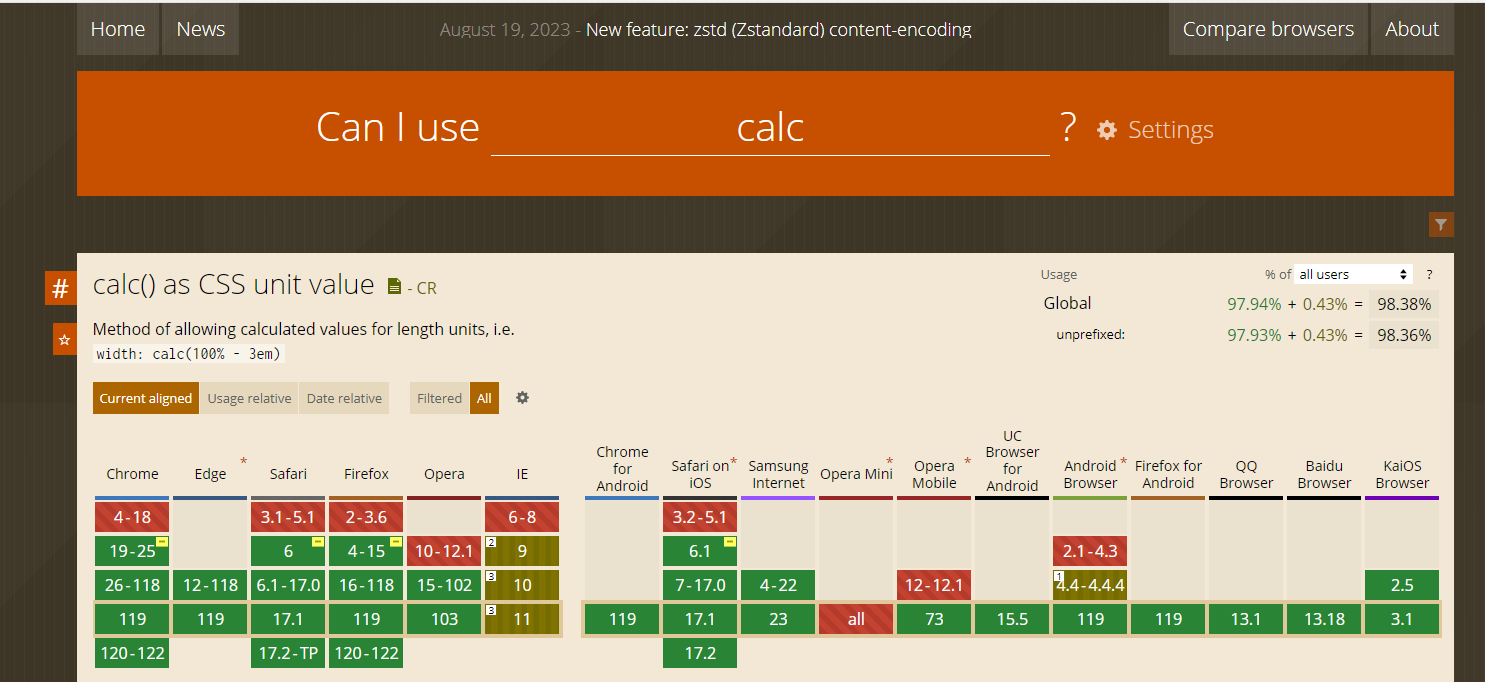
@container — это мощная новая функциональность в CSS следующего поколения, которая позволяет легко создавать адаптивные и отзывчивые контейнеры. Использование новых свойств container-type и width дает дополнительный контроль над размерами и шириной контейнеров. Она может быть использована для оформления различных элементов на веб-странице и предоставляет гибкие возможности для управления макетом и стилем. Надеюсь, что эта статья помогла вам лучше понять, что такое @container и как его использовать с примерами, включая новые свойства container-type и width.

**Использование функции calc()**

**Теория**

**calc() -** функция, позволяющая производить математические вычисления прямо в CSS. Раньше, до появления calc(), приходилось страдать и вычислять размеры примерно.

Функция поддерживается всеми современными браузерами.



**Пример**

Частая ситуация: в вёрстке три колонки, ширина каждой из которых должна быть ровно третью от 100%. 100% / 3 = 33.3333333333...%. Раньше мы допускали неточность и указывали ширину как 33%. Теперь можно использовать calc() и пусть браузер сам считает.

При отрисовке страницы браузер сам высчитает и подставит значение:

.selector {

width: calc(100% / 3);

}

Но лучше не перегружать браузер расчётами. На каждую операцию он тратит некую долю секунды и часть оперативной памяти. Если расчётов будет много, то это потенциально повлияет на скорость загрузки страницы. Используйте calc() с умом.

**Как пишется**

В круглых скобках мы можем писать любые математические операции с любыми единицами измерения, доступными в вебе (%, px, rem, em, vw, vh, vmin и т.д.). Доступны четыре стандартных операнда:

+ сложение;

- вычитание;

/ деление;

\* умножение.

Операторы сложения и вычитания обязательно с двух сторон должны отбиваться пробелом. Иначе браузер воспримет их как часть числа. Хоть операторы деления и умножения не требуют такой строгости к себе, но принято и их тоже отбивать пробелами для удобства чтения.

.selector {

width: calc(100% - 2rem);

}

Внутри скобок может быть больше одного вычисления, можно группировать операции при помощи скобок. Всё как в настоящем языке программирования. Но не стоит увлекаться: чем короче вычисление, тем проще потом его прочитать и понять, что там вообще происходит.

**Для каких свойств можно указать calc() в качестве значения? Для любых, значением которых должна быть цифра!** Причём если свойство предполагает составное значение, то можно указать функцию как часть этого значения:

.selector {

margin: calc(5vh / 4) 20px;

transition: transform calc(0.5s + 120ms);

}

Внутри круглых скобок можно складывать только числовые значения. Нельзя сложить число со строкой, хотя в полноценных языках программирования, типа JavaScript, такой трюк с лёгкостью бы удался.

Ещё одно неудачное место для этой функции — медиа выражения. Вот такая запись считается не валидной (ошибочной):

@media (min-width: calc(465px + 1vw)) {

...;

}

Хотя бы у одной из цифр внутри скобок нужно указать единицу измерения, иначе браузер не сможет понять, от чего же вести вычисление.

**Как понять**

Во время отрисовки/загрузки/рендеринга страницы браузер заглядывает в CSS и производит все вычисления из функций calc(), подставляя на их место итоговое значение. Исходя из этих значений и отрисовываются стили элементов.

Очень удобно (и часто приходится) использовать, когда из одной величины в относительных единицах надо вычесть другую величину в абсолютных единицах. Самостоятельно это никак не посчитать, а браузеру раз плюнуть. Например, каким будет результат такого вычисления?

.selector {

height: calc(100vh - 34px);

}

Допустимость использования разных единиц измерения в одном выражении, вероятно, можно назвать самой ценной возможностью calc(). Подобное применяется почти в каждом из вышеприведённых примеров. Вот, просто для того, чтобы обратить на это ваше внимание, ещё один пример, в котором показано использование различных единиц измерения.

Совместное использование процентов и пикселей: width: calc(100% - 20px). Это выражение читается так: «Ширина равна ширине элемента, из которой вычитается 20 пикселей».

Эта функция также умеет работать и с числами, применяемыми без указания единиц измерения: line-height: calc(1.2 \* 1.2).

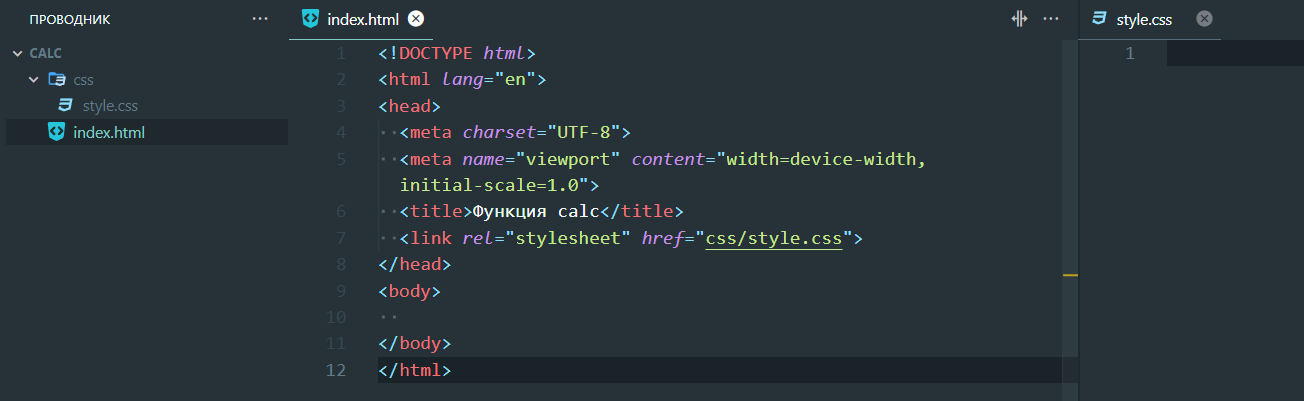
Её можно использовать и для вычисления углов: transform: rotate(calc(10deg \* 5)).

Рассмотрим данную функцию на практике.

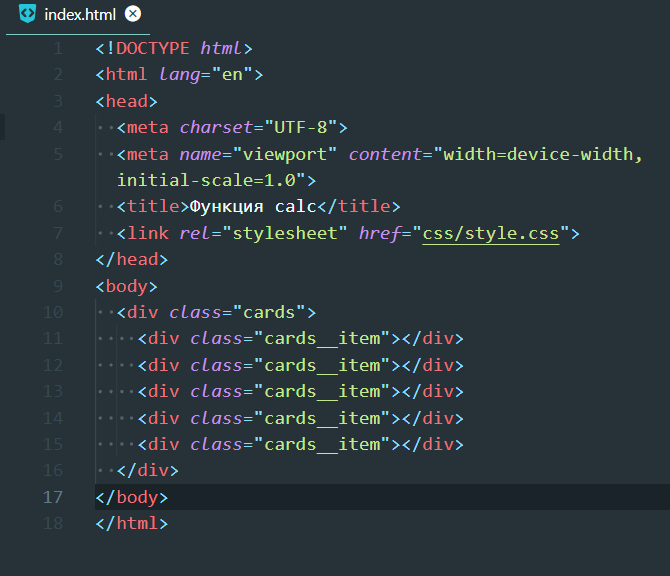
**Практика**

Рассмотрим применение расчётов на примере создания адаптивных карточек. Данный компонент очень часто встречается на разных сайтах.

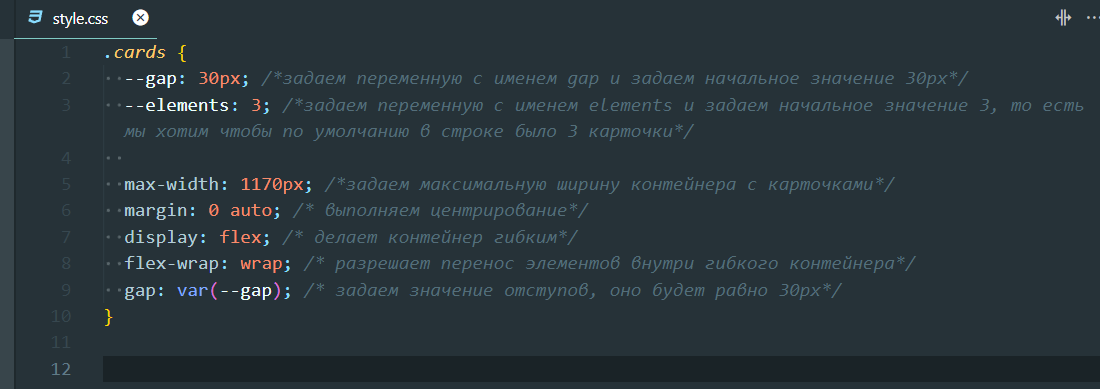
В своей папке в документах создайте папку calc и откройте ее чрез vs code. Создайте структуру проекта и скелет документа.



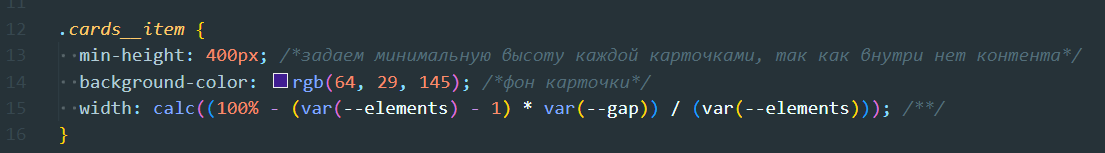
Внутри body создайте контейнер с 5 блоками. Проще всего это сделать с помощью формулы. 



Переходим в css и для контейнера зададим стили (комментарии можно не писать, если понимаете, что это за свойства и значения).



Далее напишем стили для карточек. Тут самой важной является строка 15.



**width: calc((100% - (var(--elements) - 1) \* var(--gap)) / (var(--elements)))** - этот код представляет собой вычисляемое свойство CSS для определения ширины нашей карточки.

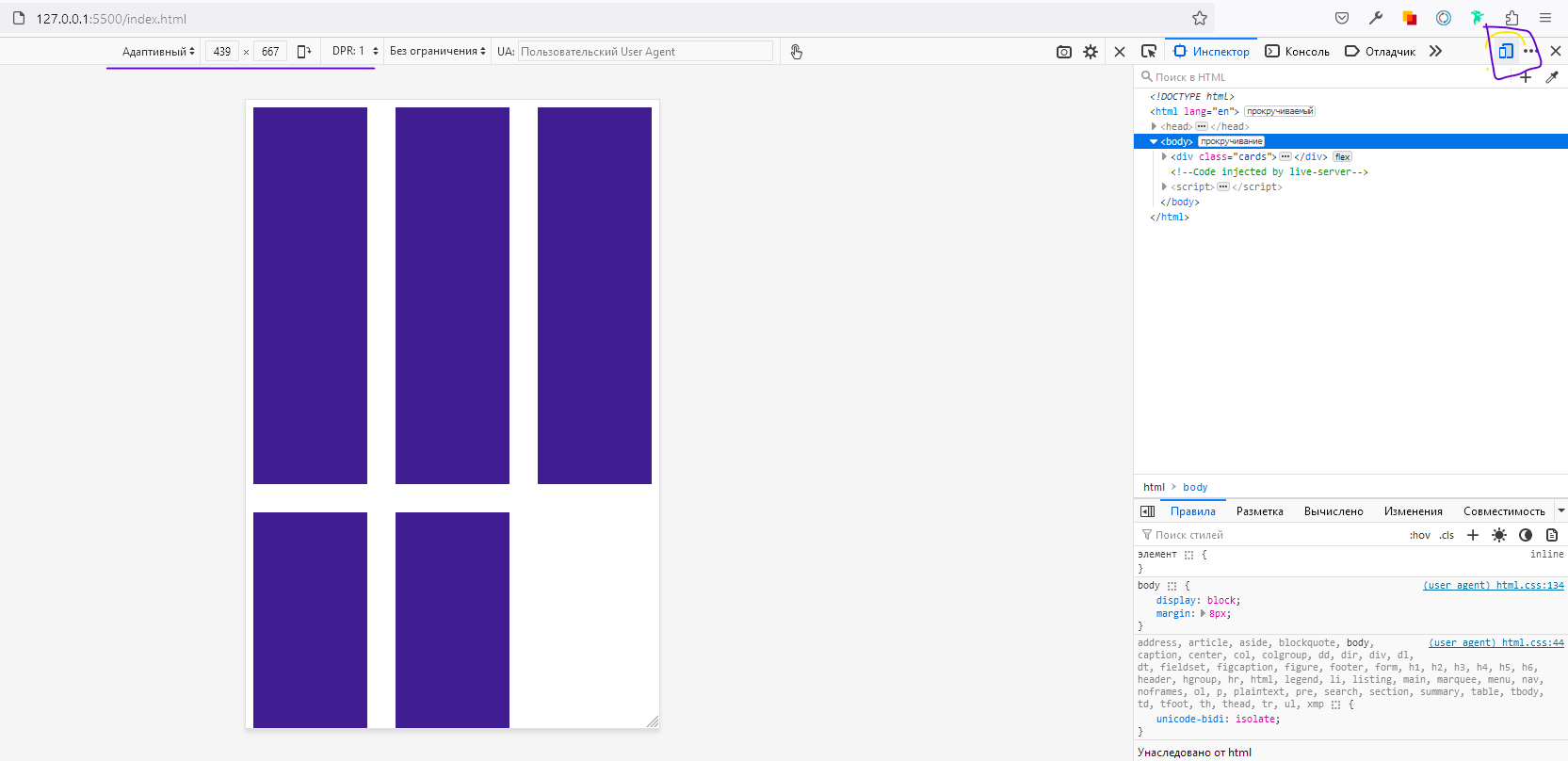
Значение этого свойства рассчитывается по формуле:

ширина = (100% - (количество\_элементов - 1) \* отступ между карточками) / количество\_элементов

* "100%" указывает на 100% ширины родительского элемента (у нас оно макисмально 1170px).
* "var(--elements)" представляет переменную CSS "--elements", которая определяет количество элементов, для которых вычисляется ширина, по умолчанию 3.
* "var(--gap)" представляет переменную CSS "--gap", которая определяет отстуммежду элементами (карточками).

Таким образом с помощью данной формулы мы можем равномерно распределить ширину элементов внутри контейнера, чтобы они занимали все доступное пространство.

Посмотрите результат в браузере. Нажмите клавишу F12 и включите режим адаптивного дизайна. Измените ширину экрана.



На маленьком экране карточки очень сильно сжимаются и, если бы там был текст или картинки, они бы стали не читабельными и странными. Необходимо реализовать дальнейший перенос элементов в зависимости от ширины экрана.

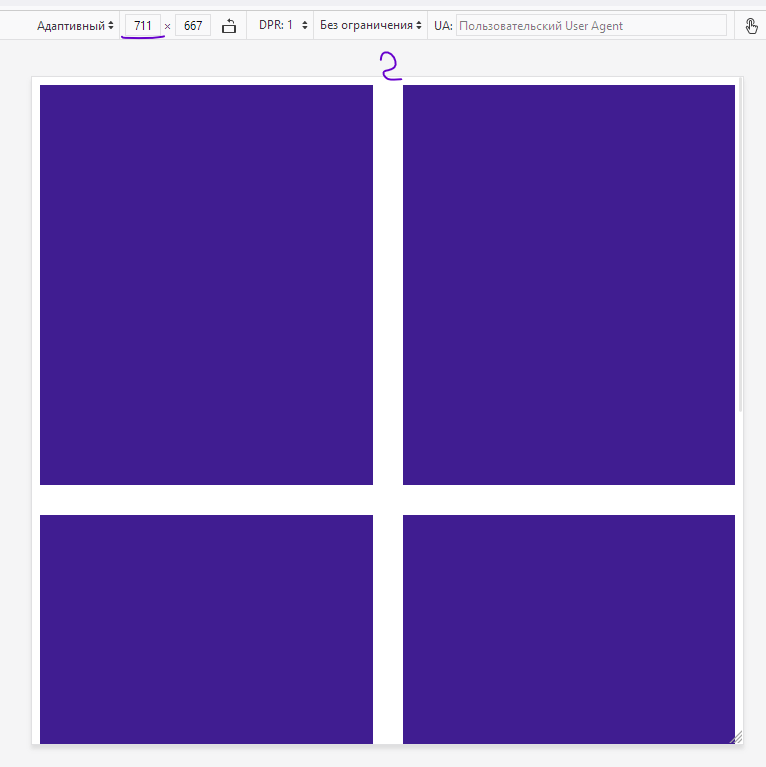
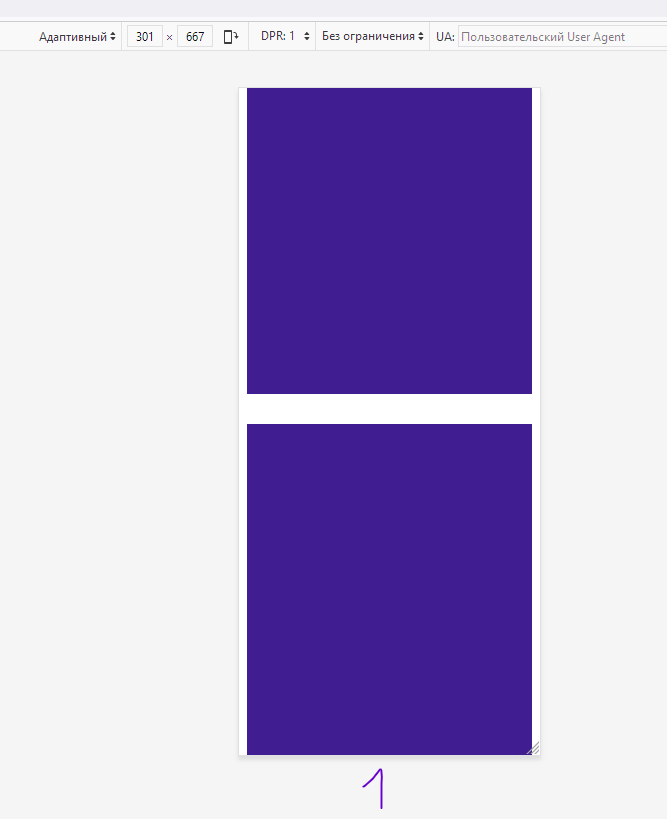
Допишем медиазапросы.



В первом медиа-запросе мы указываем, что когда ширина экрана будет меньше 1024px, то по умолчанию в ряду должно быть 2 элемента, то есть просто меняем значение переменной.

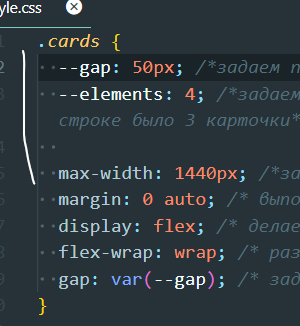
Во втором медиа-запросе мы указываем, что когда ширина экрана будет меньше 576px, то по умолчанию в ряду должно быть 1 элемент.

Снова измените ширину экрана и проверьте сработали ли медиа-запросы.

Таким образом на экранах до ширины 576 в ряду одна карточка, на экранах в диапазоне 575 -1024 – в ряду 2 карточки, на экранах больше 1024px в ряду будет 3 карточки.

Измените начальные значения и протестируете, что формула отрабатывает и карточки перестраиваются.



**Продемонстрируйте итог работы преподавателю.**

**Использование функции clamp()**

**Теория**

Чтобы улучшить адаптивность сайта, не обязательно использовать какие-то сложные слушатели событий с помощью Web API ResizeObserver, функцию calc() или media-запросы CSS, которые будут подставлять нужные элементы страницы в зависимости от размера окна браузера.

Функция clamp() задаёт предпочитаемое значение, а также его нижнюю и верхнюю границы. Это позволяет установить диапазон, в котором должно располагаться значение.

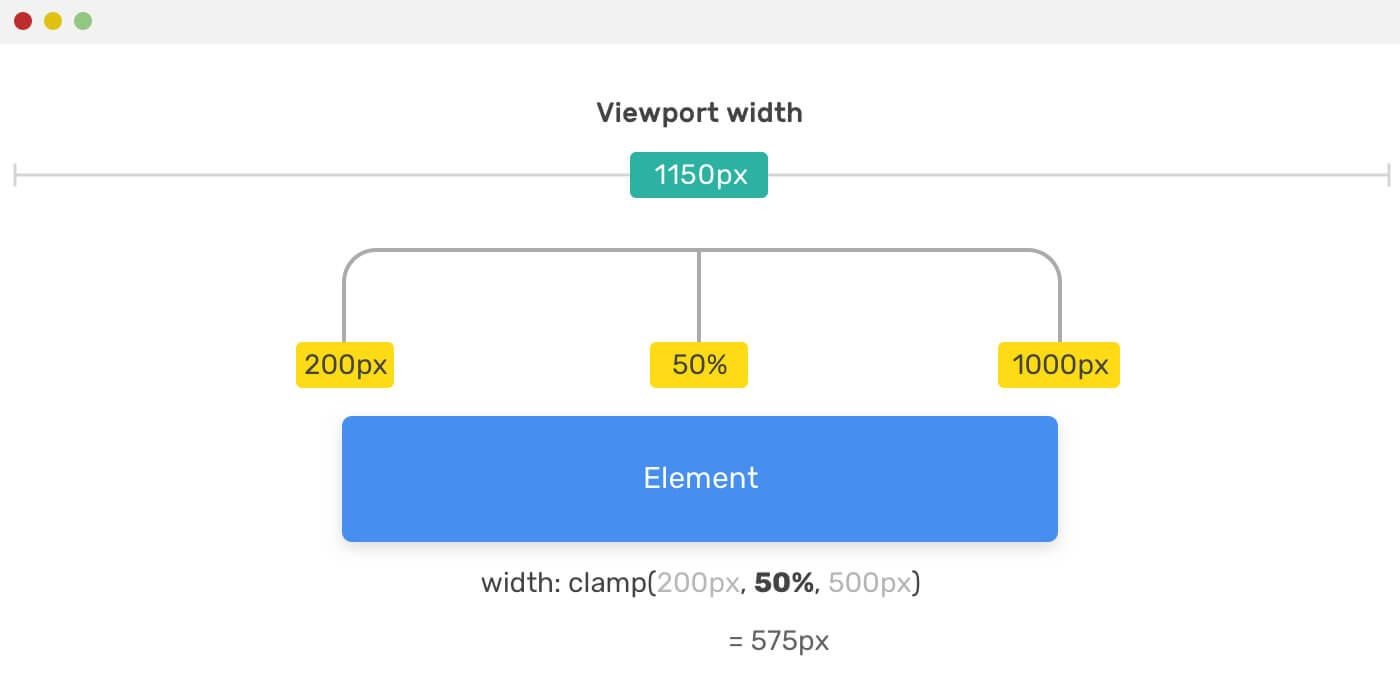
Clamp не поддерживается только в браузере Internet Explorer.

Функция clamp() позволяет ограничивать диапазон изменения некоего значения, задавая его нижний и верхний пределы.

Она принимает три параметра: минимальное значение, вычисляемое (рекомендованное) значение, максимальное значение. Если вычисляемое значение не выходит за пределы, ограничиваемые минимальным и максимальным значениями, переданными функции, то она возвратит именно это значение.

Вот пример: element {width: clamp(200px, 50%, 1000px);}

Здесь мы настраиваем ширину элемента, которая не должна быть меньше 200px и больше 1000px. Если значение в 50% не выходит за эти пределы — используется именно это значение. Вот как это выглядит.



*Если 50% ширины области просмотра попадают в диапазон 200px-1000px — для задания ширины элемента будет использоваться именно это значение. В данном случае это 575px*

Разберём этот пример:

* Ширина элемента никогда не будет меньше 200px.
* Второй параметр функции, заданный как 50%, будет использоваться только в том случае, если ширина области просмотра будет больше 400px и меньше 2000px.
* Ширина элемента не превысит 1000px.

В итоге можно сказать, что функция clamp() позволяет задавать диапазон, в котором может изменяться передаваемое ей вычисляемое значение.

### **Как вычисляются результаты функции clamp()?**

На [MDN](https://developer.mozilla.org/en-US/docs/Web/CSS/clamp) можно найти сведения о том, что, когда в качестве значения CSS-свойства используется функция clamp(), она является эквивалентом конструкции, в которой применяются функции max() и min().

Взгляните на следующий пример:

.element {

width: clamp(200px, 50%, 1000px);

/\* Это - эквивалент следующего выражения \*/

width: max(200px, min(50%, 1000px));

}

Значение 50% зависит от ширины области просмотра браузера. Представим, что ширина области просмотра составляет 1150px.

.element {

width: max(200px, min(50%, 1000px));

/\* Представим, что ширина области просмотра составляет 1150px \*/

width: max(200px, min(575px, 1000px));

/\* Промежуточный результат \*/

width: max(200px, 575px);

/\* Итоговый результат \*/

width: 575px;

}

### **Контекст имеет значение**

Вычисленное значение зависит от контекста. При указании вычисляемого значения можно использовать различные единицы измерения: %, em, rem, vw/vh. Даже если значение выражено в процентах, при его расчёте может использоваться либо ширина области просмотра — в том случае, если элемент находится в теге <body>, либо ширина другого элемента, являющегося контейнером элемента.

### **Математические выражения**

Стоит сказать о том, что при использовании функции clamp() можно передавать ей математические выражения, что избавляет нас от необходимости использования функции calc(). Спецификация говорит нам о том, что в каждом из аргументов clamp() можно использовать полные математические выражения. Поэтому здесь нет нужды во вложенных функциях calc(). Кроме того, если к итоговому значению нужно применить несколько ограничений, функции можно передавать более двух аргументов.

Взгляните на следующий пример:

.type {

/\* Ограничиваем font-size значениями, находящимися между 12px и 100px \*/

 font-size: clamp(12px, 10 \* (1vw + 1vh) / 2, 100px);

}

Отличным сценарием использования функции clamp() является гибкая настройка размера шрифта заголовков. Представим, что нам нужно, чтобы минимальным размером шрифта заголовка было бы 16px, а максимальным — 50px. Благодаря использованию функции clamp() мы можем сделать так, чтобы размер шрифта находился бы в пределах этих значений, не уходя ни ниже, ни выше.

.title {

font-size: clamp(16px, 5vw, 50px);

}

Функция clamp() подходит для ограничения размеров внутренних отступов разделов страниц. Взгляните на следующий пример, в котором показана настройка верхней части страницы (hero-раздела).



Гибко настроить отступы подобного раздела можно, воспользовавшись всего одной строкой CSS-кода.

.hero {

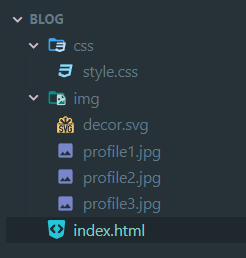
padding: clamp(2rem, 10vmax, 10rem) 1rem;

}

**Практика**

Применим на практике функцию и свойство @container. В данном проекте мы применим подход mobile-first, то есть сначала делаем версию для мобилки и потом изменяем внешний вид страницы в зависимости от увеличения ширины экрана. Цель работы понять, как применять функции, поэтому не будем заморачиваться над дизайном.

В своей папке в документах создайте папку blog и откройте ее чрез vs code. Создайте структуру проекта и скелет документа. Скопируйте картинки в папку img.



Далее выполним подключение дополнительного файла для нормализации стилей.

Normalize.css – это небольшой CSS-файл, который обеспечивает для HTML-элементов лучшую кроссбраузерность в стилях по умолчанию. Это современная, готовая к HTML5 альтернатива традиционному сбросу стилей reset.css.

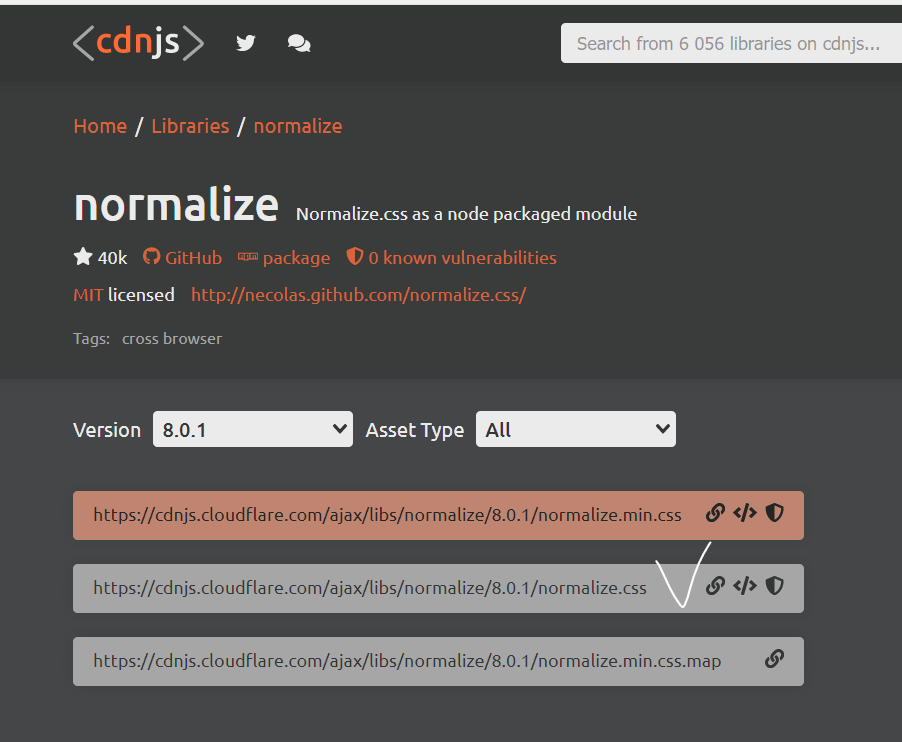
Цели normalize.css:

* сохранять полезные настройки браузера, а не стирать их;
* нормализовать стили для широкого круга HTML-элементов;
* корректировать ошибки и основные несоответствия браузера;
* совершенствовать юзабилити незаметными улучшениями;
* объяснять код, используя комментарии и детальную документацию.

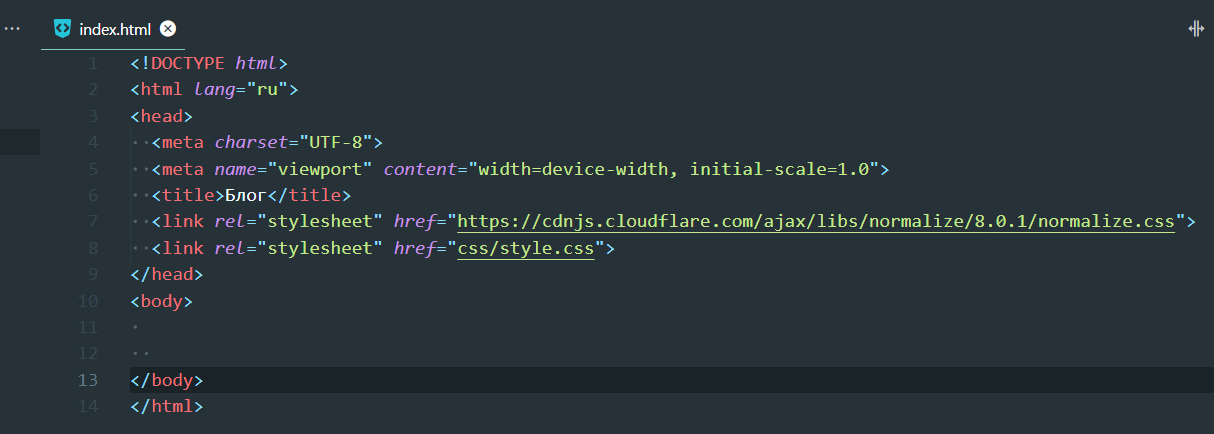
Он поддерживает широкий диапазон браузеров (в том числе мобильных) и включает в себя CSS, который нормализует HTML5-элементы, типографику, списки, встраиваемый контент, формы и таблицы.

Несмотря на то, что проект основан на принципе нормализации, он использует стандартные настройки там, где они предпочтительны.

Для подключения файла переходим на сайт <https://cdnjs.com/libraries/normalize> и копируем ссылку.

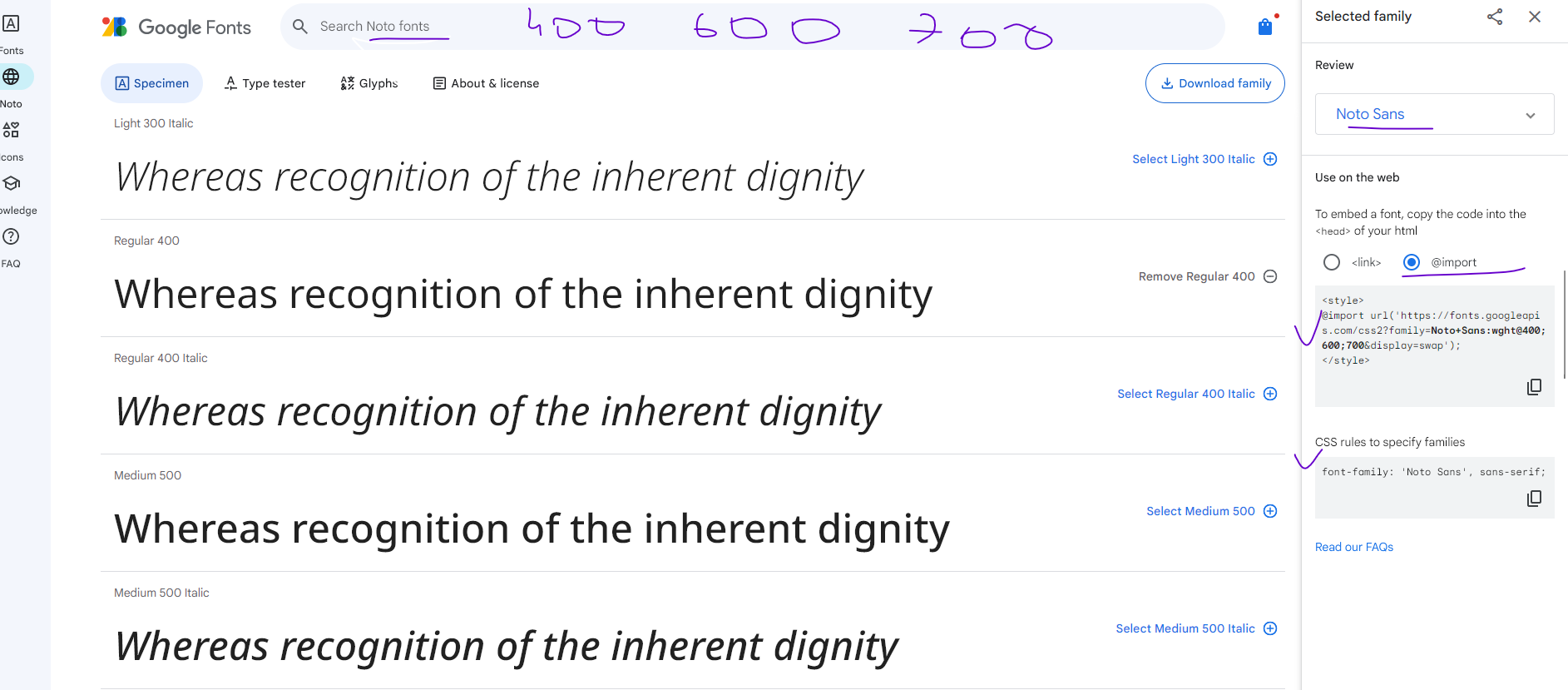


Далее делаем стандартное подключение css и вставляем ссылку.

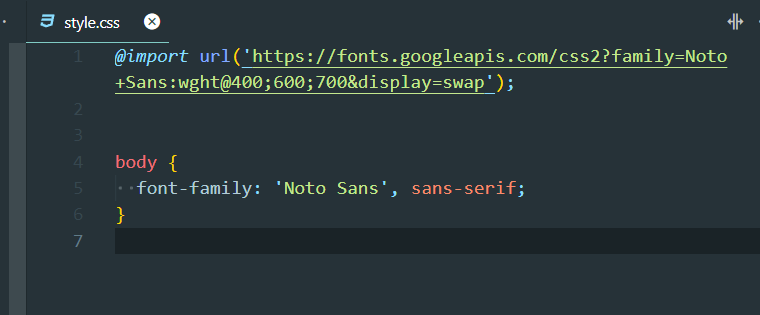


Выполним подключение шрифта. Переходим на сайт <https://fonts.google.com/>

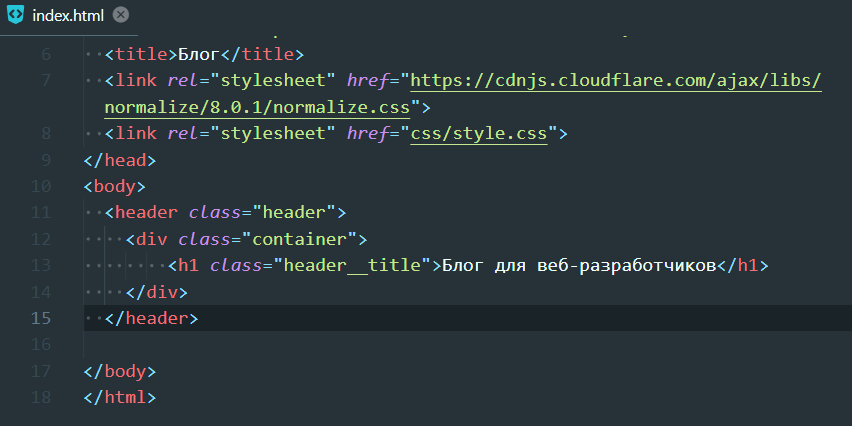
Нам нужен шрифт Noto Sans – 400,600,700.



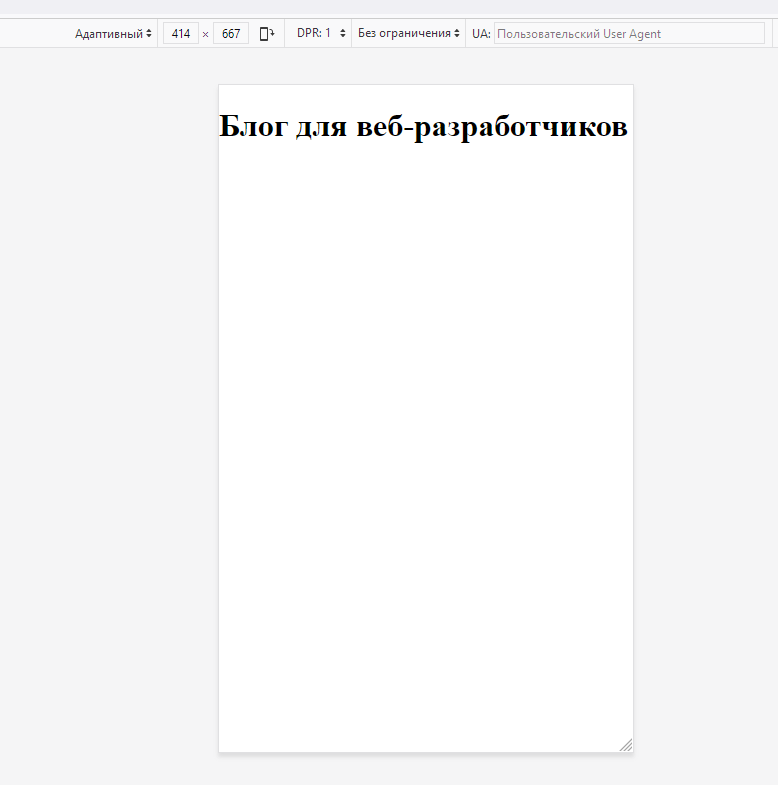
Подключение сделаем в css.



Сверстаем шапку и добавим в нее заголовок. По макету видно, что в шапке есть ограничения, поэтому добавляем ограничивающий контейнер.

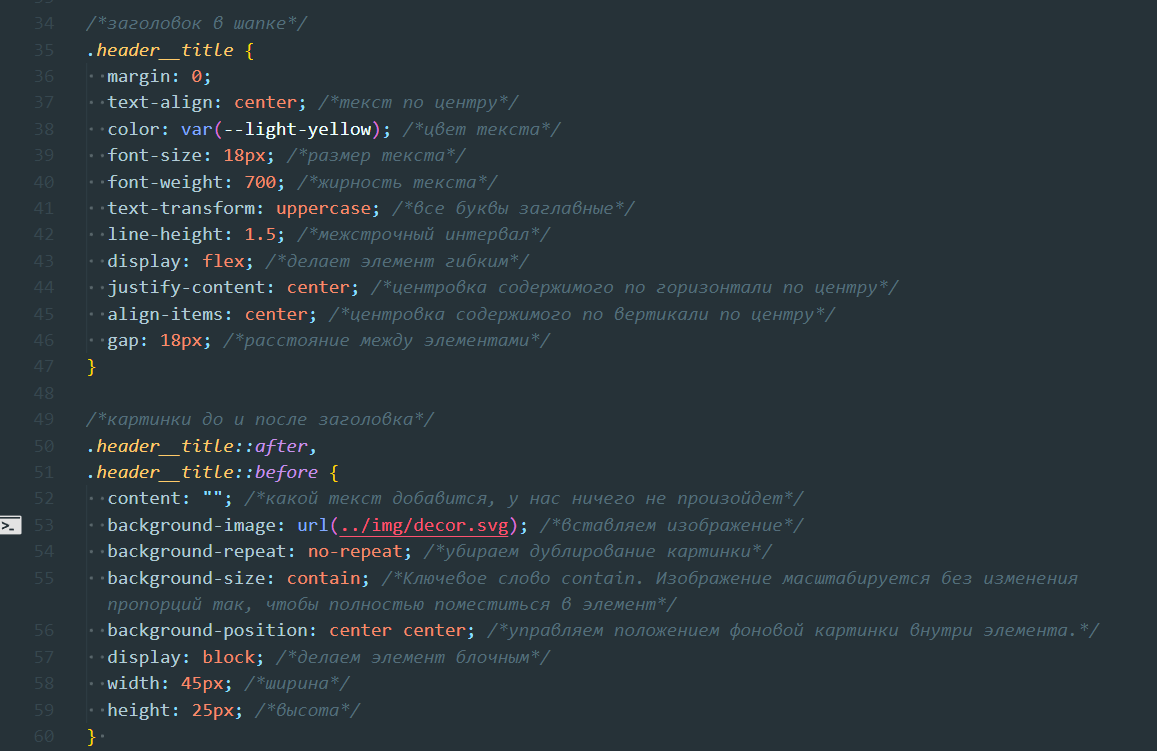


Запустите просмотр в браузере Live Server и сделайте экран маленьким. Напоминаю, что сначала мы делаем верстку под смартфон.



Стилизуем добавленные элементы.

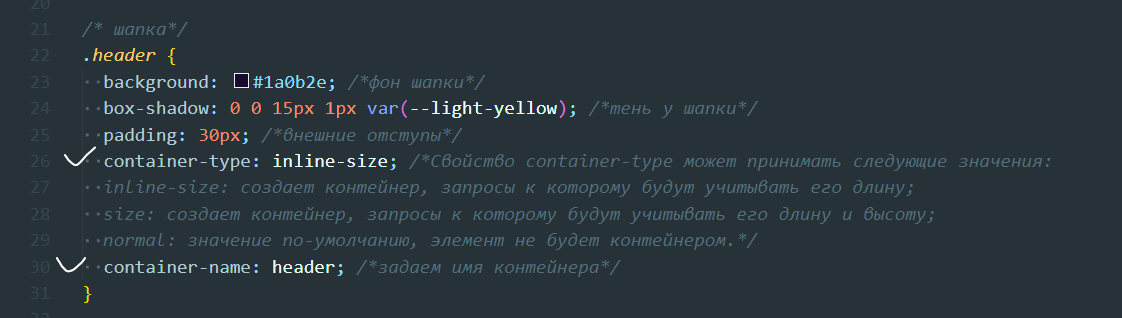




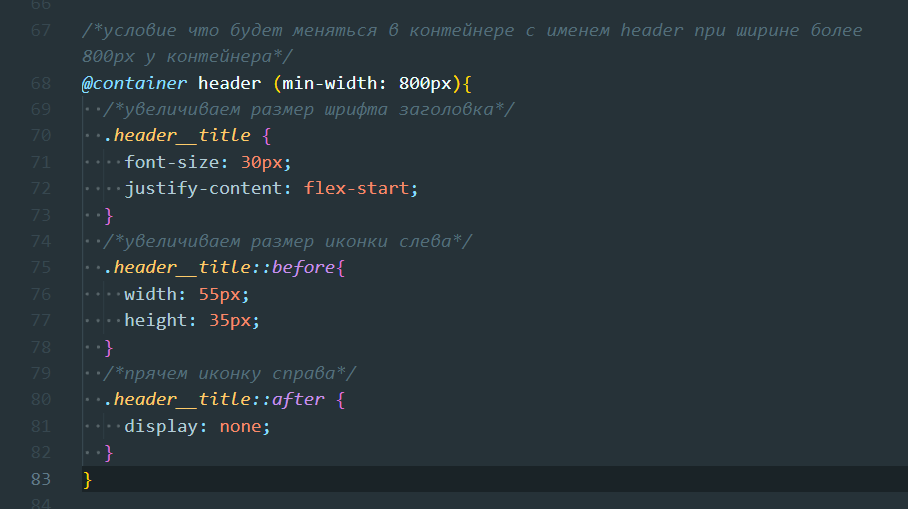
Далее применим на практике свойство @container.

В данном случае сделаем контейнером саму шапку и будем управлять элементами внутри нее в зависимости от ширины экрана и самого контейнера.

Допишем свойства для шапки.



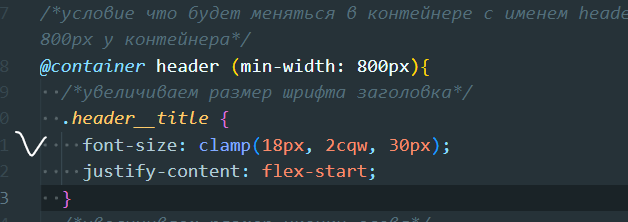
Далее нам нужно реализовать увеличение размера шрифта и картинок в зависимости от ширины экрана, а также на большом экране убрать картинку справа. Помимо этого, на большем экране заголовок размещается уже не по центру, а слева.



Изменяйте ширину экрана, чтобы отслеживать изменения в браузере.

Все норм, но свойства меняются слишком резко. Фиксированные значения в px не являются адаптивными, поэтому часть кода нужно переделать и улучшить.

Добавим для заголовка расчет размера по функции.



Функция clamp() является новым свойством CSS и позволяет установить пределы для значения свойства.

В данном коде clamp(18px, 2cqw, 30px):

18px - это минимальное значение,

30px - максимальное значение,

2cqw - это предпочтительное значение.

Новые единицы измерения cqw и cqh задают длину относительно размеров контейнера.

cq в начале - это сокращение от английского "query container". Третья буква после (w и h) сообщают нам, относительно чего будут происходить изменения.

cqw — это 1% от ширины контейнера

cqh — 1% от высоты контейнера

Также зададим размер декоративных элементов с помощью функции.

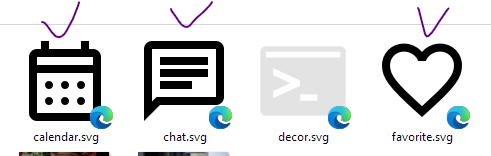


Протестируйте плавность изменения текста и декоративных элементов.

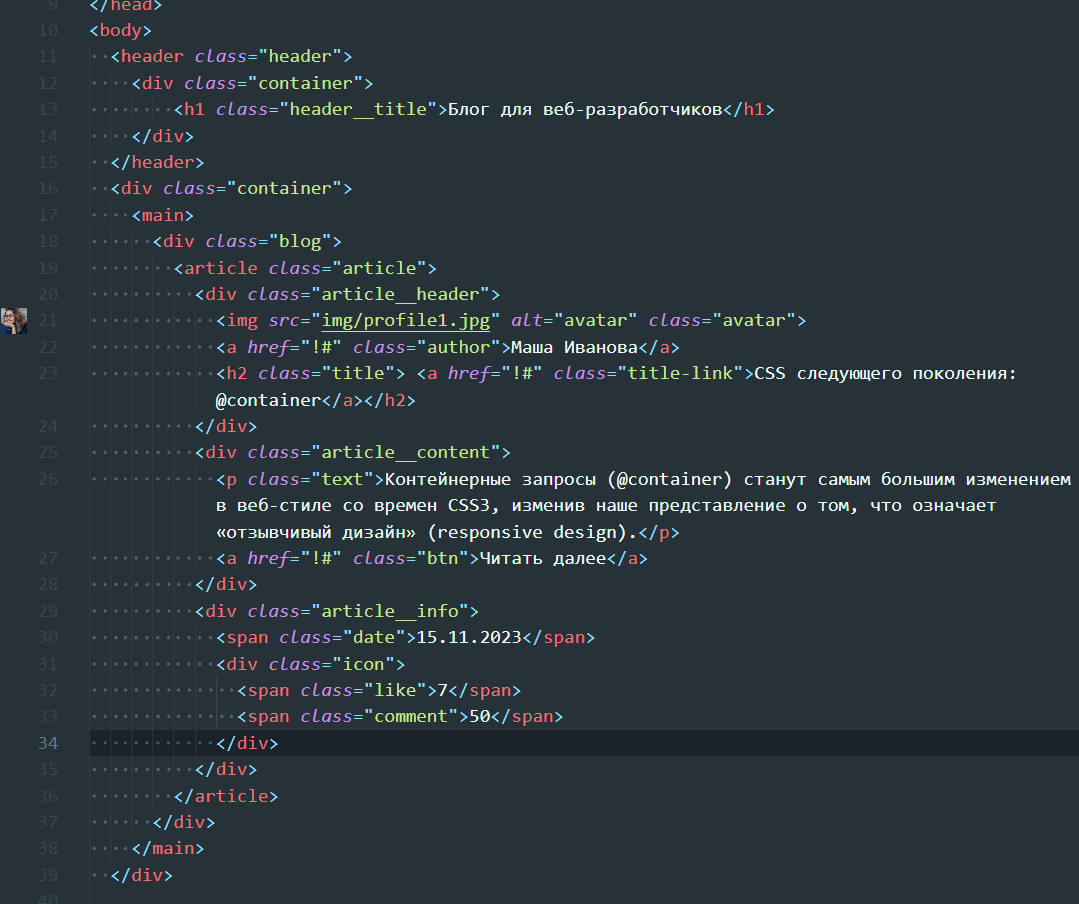
**Статьи**

Сделаем разметку и оформление статьи.

Для начала перейдите на сайт <https://fonts.google.com/icons> и скачайте в формате svg иконки.



Делаем разметку для блога и первой статьи. Мы делаем верстку сначала для смартфона, поэтому в браузере не забывайте включать режим адаптивного дизайна (375px).



Текст для вставки:

Маша Иванова

CSS следующего поколения: @container

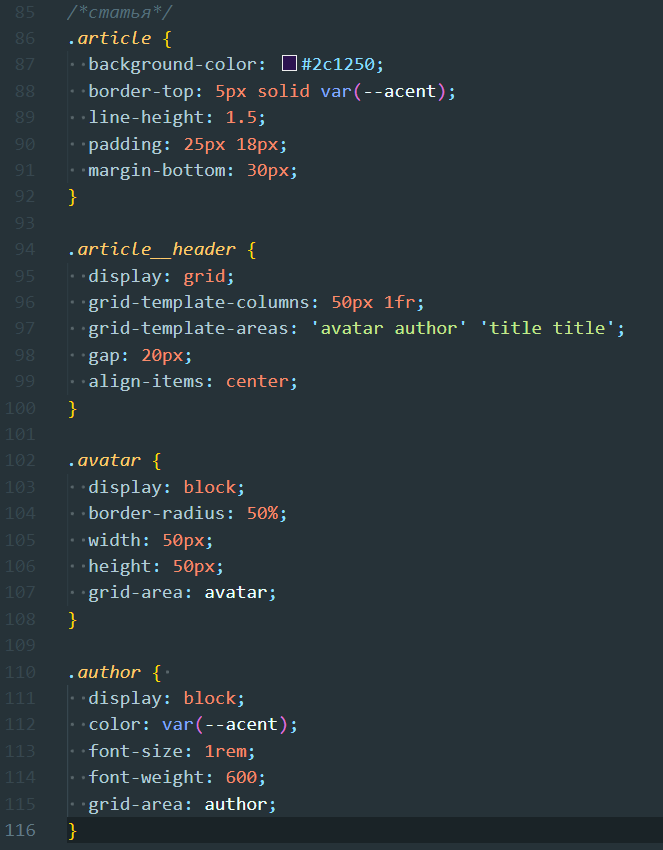
Контейнерные запросы (@container) станут самым большим изменением в веб-стиле со времен CSS3, изменив наше представление о том, что означает «отзывчивый дизайн» (responsive design).

Читать далее

15.11.2023

7 50

Далее сделаем оформление статьи и ее элементов. Тут применяется грид- сетка.



Этот код определяет стили для заголовка статьи, используя CSS Grid.

display: grid; задает элементу .article\_\_header использование CSS Grid для размещения его дочерних элементов.

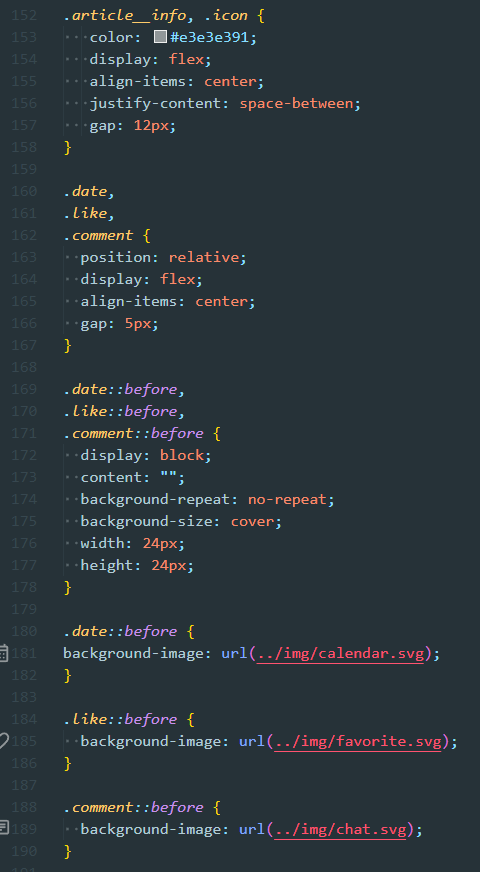
grid-template-columns: 50px 1fr; устанавливает ширину колонок в сетке. Первая колонка будет иметь ширину 50 пикселей, а вторая колонка будет занимать оставшееся пространство (1fr).

grid-template-areas: 'avatar author' 'title title'; определяет расположение дочерних элементов .avatar, .author и .title в сетке. В данном случае, .avatar и .author находятся в одной строке, а .title находится на следующей строке.

gap: 20px; задает промежуток (зазор) между элементами в сетке.

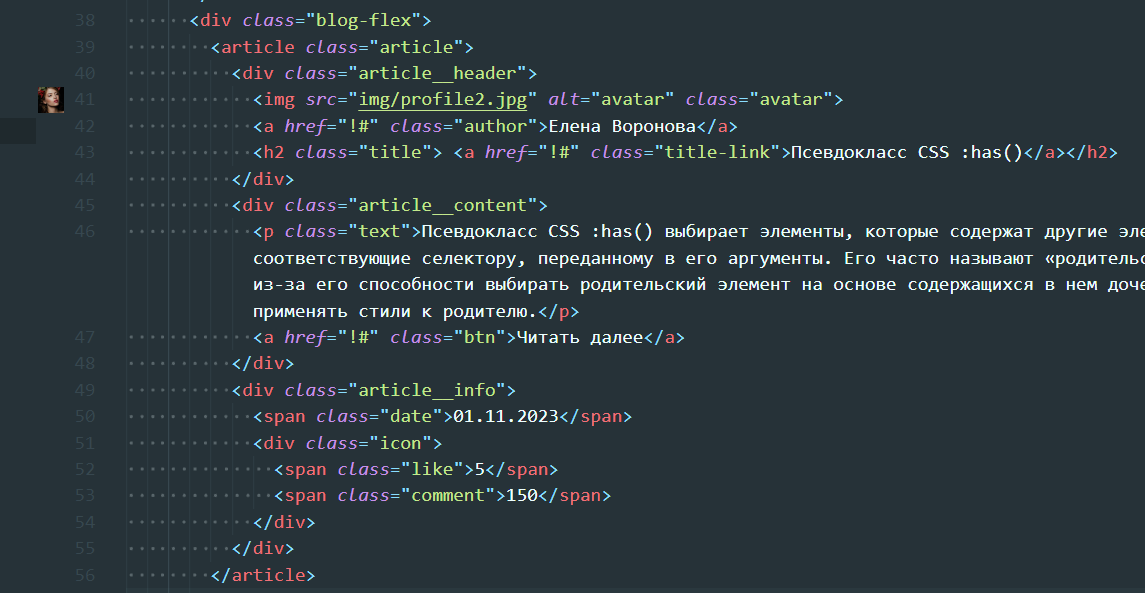
align-items: center; выравнивает элементы по вертикали.

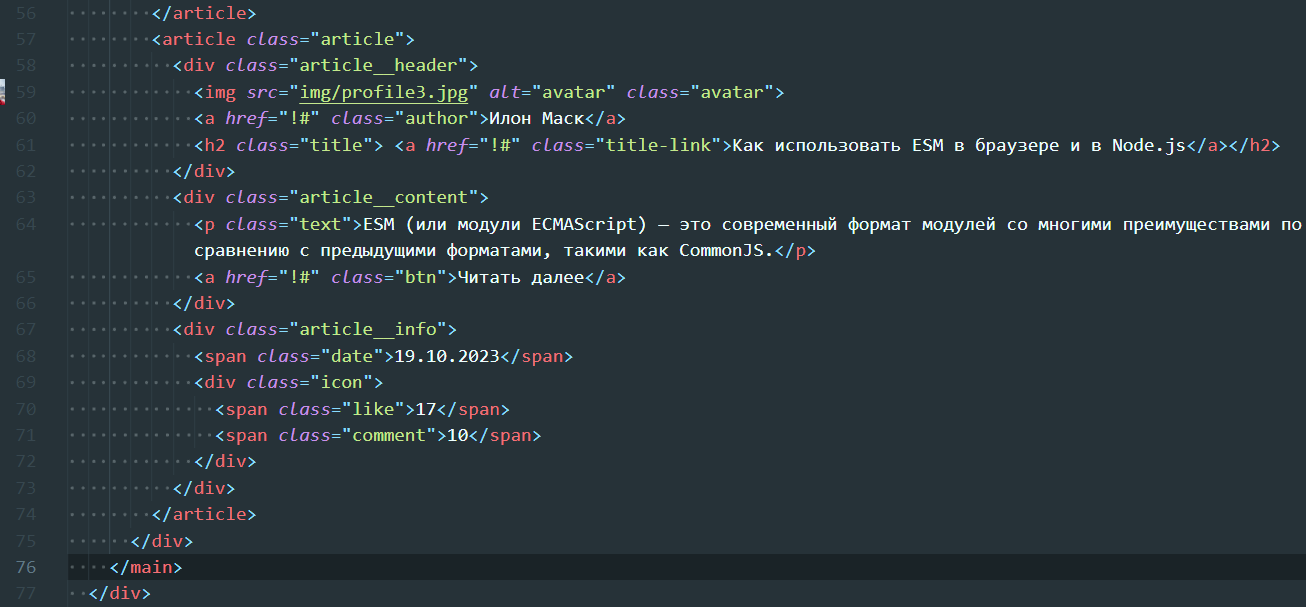
grid-area: head; определяет имя области сетки, к которой будет принадлежать элемент .article\_\_header. Это имя может быть использовано в родительском элементе для размещения других элементов в этой области.

Далее создаем блок, в котором на больших экранах статьи будут в 2 колонки и дублируем статьи, меняем контент.







Текст для вставки:

Елена Воронова

Псевдокласс CSS :has()

Псевдокласс CSS :has() выбирает элементы, которые содержат другие элементы, соответствующие селектору, переданному в его аргументы. Его часто называют «родительским селектором» из-за его способности выбирать родительский элемент на основе содержащихся в нем дочерних элементов и применять стили к родителю.

Читать далее

01.11.2023

5 150

Илон Маск

Как использовать ESM в браузере

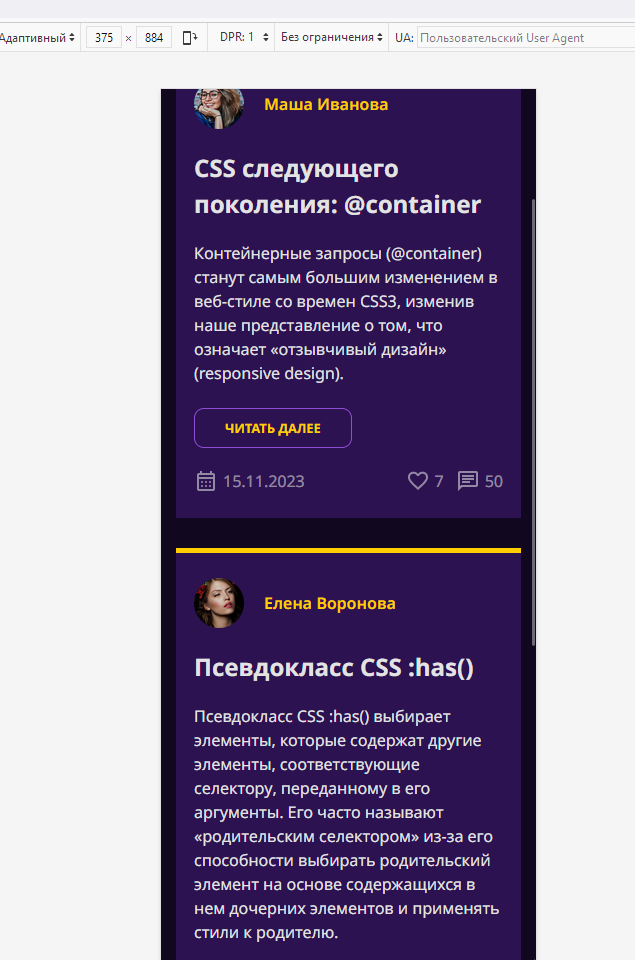
ESM (или модули ECMAScript) — это современный формат модулей со многими преимуществами по сравнению с предыдущими форматами, такими как CommonJS.Модули определяются с помощью различных операторов импорт и экспорт.

Читать далее

19.10.2023

17 10

В результате у вас должен был получиться результат:



Далее применим контейнер. В данном случае контейнером будет статья.



Далее задаем параметры для контейнера и его содержимого для среднего размера экрана.



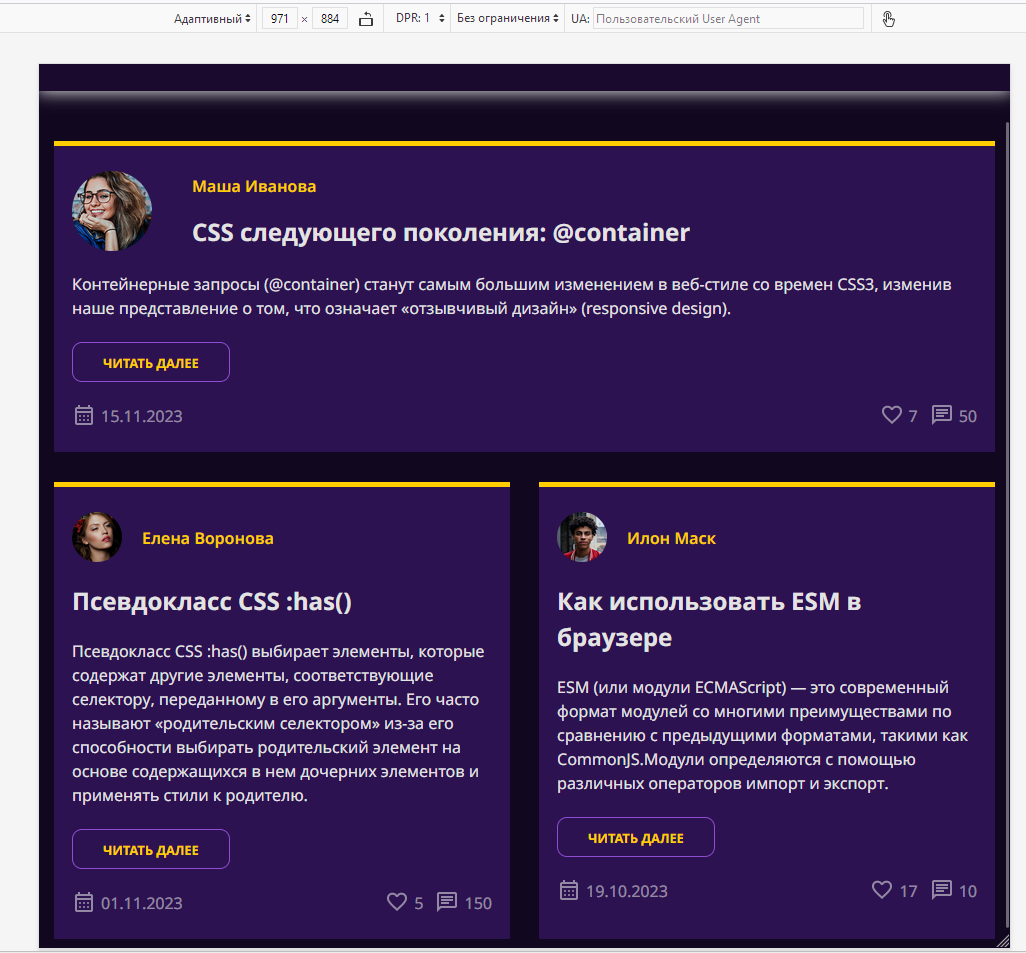
Единица ch определяет ширину нулевого символа (0). Если браузер не может вычислить размеры этого символа (к примеру, для пользовательского шрифта), тогда ширина символа принимается равной 0.5em, а высота 1em.

Для моноширинного шрифта (например, Courier), в котором ширина всех символов одинакова, 1ch равен ширине одного произвольного символа. Для остальных шрифтов (Times, Helvetica и др.) любой символ может оказаться у́же или шире нуля.

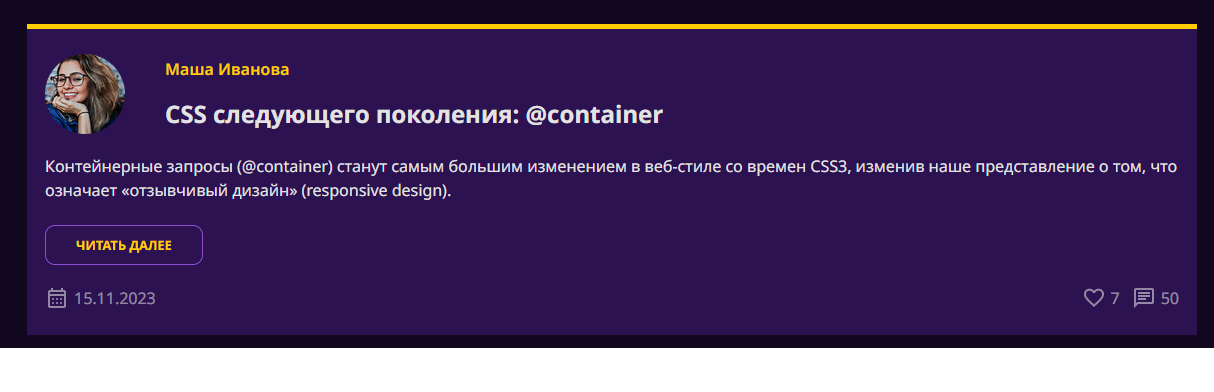
Переделаем контент в 2 колонки. Перед контейнером допишем правила.



Чтобы проверить результат, увеличьте ширину экрана, ведь мы сейчас делаем внешний вид для среднего экрана.



**Самостоятельная работа:** на больших экранах внутри первой карточки контент разбегается и выглядит не очень красиво. Сделайте перестройку контента на десктопе, чтобы данная карточка выглядела красиво.



Улучшите внешний вид данной карточки.