**CSS Grid**

Технология CSS Grid родилась в 2018м году и довольно быстро завоевала популярность благодаря своему удобству и исправлением проблем от других способов.

CSS Grid скоро станет неотъемлемой частью набора инструментов любого фронтэнд разработчика. И если вы один из них, то вам придется учить CSS Grid — который уже точно станет неоспоримо важным умением для любой позиции в фронтэнд разработке.

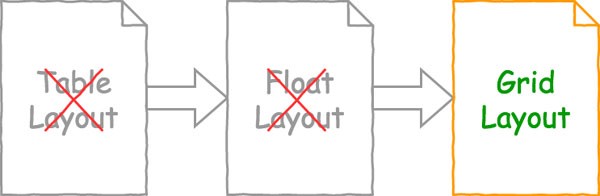
**Введение**

CSS Grid это новая модель для создания шаблонов, оптимизированная для создания двумерных макетов. Она идеально подходит для: *шаблонов сайтов, форм, галерей и всего, что требует точного и отзывчивого позиционирования.*

С развитием веба, в последние годы, разработка шаблонов веб-сайтов стала всё более затруднительной. В ранние годы веба, HTML таблицы зачастую использовались для многоколоночных шаблонов, форм и т.д. Но у этого метода есть свои недостатки. По факту, таблицы были созданы для того, чтобы содержать табличные данные и только для этого. И уже даже не касаясь семантических проблем, табличные шаблоны совершенно не созданы для адаптивного дизайна.

*Флоаты* в конечном итоге заменили таблицы как всеобще приемлемый метод создания разметки шаблона, так как он позволяет нам позиционировать элементы вне зависимости от изначальной разметки. Тем не менее, хотя этот метод считался значительным улучшением на фоне табличной верстки, у него тоже были свои ограничения. *Флоаты* в основном были разработаны для создания шаблонов документов и они не совсем были подходящими для создания сложных шаблонов приложений, которые сейчас распространены в сети и которые мы привыкли видеть в своих браузерах. *Флоат элементы* сложно контролировать, особенно на устройствах разных размеров. Это привело к различным *гридоподобным* хакам, которые уже стали нормой, большинству из них требовалась дополнительная разметка, которая отвлекала от общей концепции разделения контента. Таким образом рабочая группа CSS находилась в поиске наилучшего решения для всех задач.

CSS Grid модель рассматривает все эти проблемы и даже более. Она позволяет нам создавать продвинутые шаблоны за малую долю времени от того, которое бы вы потратили на них с флотами и с гораздо меньшим количеством кода.



Хотя flexbox уже позволил разработчикам начать уходить от флоат элементов, но он работает только в одном измерении. Grid CSS же это делает в двух, таким образом лучше подходя для создания сложных, комплексных шаблонов.

Вторая важная "фишка" гридов - они избавляют от головной боли и хаков в решении первоочередной задачи верстки страниц — размещения и взаиморасположения основных структурных блоков (шапка, меню, сайдбар, подвал и т.п.). До сих пор для этого требовались горы хаков. Даже флексбоксы, хотя и здорово облегчили задачу, не решили ее до конца — с ними по мере загрузки страница перестраивается, блоки перепрыгивают, удаление/добавление блока влияет на поведение остальных ит.д. С гридами же задача решается тривиально, а блоки остаются каждый на своем месте, надежность старых добрых таблиц и гибкость семантической разметки «в одном флаконе».

**Как это работает?**

Grid шаблон работает по системе сеток. Grid это набор пересекающихся горизонтальных и вертикальных линий, которые создают размерность и позиционируют систему координат для контента в самом gridконтейнере.

Чтобы создать Grid разметку, вам просто нужно выставить элементу display: grid. Этот шаг автоматически сделает всех прямых потомков этого элемента — grid элементами. После этого вы можете смело использовать разнообразные grid свойства для выравнивания размеров и позиционирования элементов должным образом.

**Основные термины**

* **Grid container**
* **Grid lines**
* **Grid track**
* **Grid cell**
* **Grid area**
* **Grid gutter**

Каждый элемент тесно связан друг с другом и отвечает за определенную часть grid контейнера.

Пример:

<div class="container">

<div class="item item1"></div>

<div class="item item2"></div>

<div class="item item3"></div>

<div class="item item4"></div>

<div class="item item5"></div>

<div class="item item6"></div>

</div>

Добавляем CSS:

.container {

display: grid;

grid-template-columns: 150px 150px 150px;

grid-template-rows: 150px 150px;

gap: 10px;

}

Наш результат:

**Единицы длины**

Для создания треков (колонок и столбцов) можно использовать любые единицы измерения длины или проценты. Если сумма ширин треков меньше доступной в grid-контейнере, по умолчанию треки выстраиваются в начале контейнера, а в конце остаётся свободное место. Это происходит из-за того, что по умолчанию свойства align-content и justify-content имеют значение start, но их можно менять на другие уже знакомые нам значения.

Также можно использовать ключевые слова min-content, max-content и fit-content().

Использование min-content даст вам трек, который будет настолько маленьким, насколько это возможно, при этом не вызывая переполнение содержимым. Следовательно, при его использовании в качестве размера колонки, содержимое будет мягко переноситься в тех местах, где это возможно. Трек обретает ширину, равную самому длинному слову в колонке или наибольшему элементу фиксированного размера.

Использование max-content приведёт к тому, что содержимое вообще не будет переноситься, что может привести к переполнению.

Ключевое слово fit-content может быть использовано только вместе с передаваемым значением. Это значение становится максимумом, до которого может растягиваться данный трек. Следовательно, трек будет вести себя как при использовании max-content, растягивая содержимое, пока ширина не достигнет максимального значения, которое мы указали. А когда это произойдёт, содержимое начнёт переноситься как обычно. Таким образом, трек может быть меньше переданного значения, но никогда не будет больше него.

Если в итоге у вас получились треки, которые занимают больше пространства, чем доступно в контейнере, они будут переполнять контейнер. Если используются проценты, то как и в основанной на процентах float или flex разметке, нужно будет внимательно следить за тем, чтобы сумма процентов не превышала 100, если вы хотите избежать переполнения.

**Единицы измерения «fr»**

CSS Grid включает метод, который может помочь сэкономить время на расчет процентов, этот метод предполагает задание ширины треков с помощью единиц измерения fr. Это гибкая единица измерения, которая рассчитывается исходя из количества доступного пространства в grid-контейнере.

Это значит, что задав список треков 1fr 1fr 1fr, доступное пространство делится на 3 и распределяется поровну между тремя колонками. Если же указать список треков 2fr 1fr 1fr, доступное пространство делится на 4, из которых 2 части отходят первой колонке, и по одной части для второй и третьей колонки.

Важно помнить о том, что по умолчанию "доступное пространство" – это не всё пространство контейнера. Если какая-то из колонок содержит элемент фиксированного размера или длинное слово, которое не может быть перенесено на новую строку, пространство под эту колонку будет выделено до того, как начнётся его распределение между всеми колонками.

Вы можете объединять единицы измерения fr с фиксированными размерами. Например, компонент может состоять из двух колонок с фиксированной шириной по краям и гибкой центральной областью.

Использование fr совместно со свойством fit-content – это способ создавать очень гибкие компоненты сайта.

**Функция repeat()**

Функция repeat() при указании списка треков может избавить от необходимости снова и снова вводить одинаковые значения.

grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr;

grid-template-columns: repeat(12, 1fr);

При использовании функции repeat(), значение перед запятой задаёт количество повторений для списка треков, идущих после запятой. Этот список треков может содержать множество значений.В итоге, вы можете задавать таким образом повторение определённого набора треков.

Функцию repeat() можно использовать лишь для части треков из всего набора. Например, следующая строка задает одну колонку шириной 1fr, затем три колонки по 200px и в конце еще одну шириной 1fr.

grid-template-columns: 1fr repeat(3,200px) 1fr

Пример для того же набора блоков:

.container {

display: grid;

width: 800px;

grid-template-columns: repeat(3, 1fr);

grid-template-rows: repeat(2, 150px);

gap: 1rem;

}

Результат:

**Выравнивание**

Свойства justify-items и justify-self выравнивают элементы по строке, а align-items и align-self выравнивают по колонкам относительно границ ячейки. Свойства принимают значения почти аналогично flex:

* start
* end
* center
* stretch

Свойства justify-content и align-content выравнивают элементы по строке и по колонкам относительно grid container. Свойства принимают значения:

* start
* end
* center
* stretch
* space-between
* space-around
* space-evenly

**Именованные линии**

Мы с вами уже изучили, что в гридах можно задвать именованные области, но на самом деле можно создавать индивидуальные имена даже для обычных линий.

Имена линиям можно задать внутри квадратных скобок в списке треков. Здесь важно помнить, что вы именуете линию, а не трек, который следует после неё. Назвав линии, вы можете использовать их имена вместо номеров при позиционировании элементов на сетке.

.grid {

display: grid;

gap: 20px;

grid-template-rows: [header-start] 20vh [header-end] ;

grid-template-columns: [sidebar-start] 1fr 2fr [sidebar-end] 1fr 2fr 1fr 2fr;

}

.header {

grid-row: header-start;

grid-column: 1 / -1;

}

.sidebar {

grid-row: 2;

grid-column: sidebar-start / sidebar-end;

}

Имя может быть любым, кроме ключевого слова span.

Хорошей практикой является именование с использованием суффикса -start для начальных линий (не важно, это линия строки или колонки), и -end для конечных. В итоге должно получиться main-start и main-end или sidebar-start и sidebar-end

Довольно часто конечная линия одной части сетки и начальная линия другой части совпадают, но это не проблема, так как линии могут иметь несколько имён. Задать линии несколько имён можно, добавив их через пробел внутри квадратных скобок.

.grid {

display: grid;

gap: 20px;

grid-template-rows: [header-start] 20vh [header-end] ;

grid-template-columns: [full-start sidebar-start] 1fr 2fr [sidebar-end main-start] 1fr 2fr 1fr 2fr [main-end full-end];

}

.header {

grid-row: header-start;

grid-column: full-start / full-end;

}

.sidebar {

grid-row: 2;

grid-column: sidebar-start / sidebar-end;

}

.content {

grid-row: 2;

grid-column: main-start / main-end;

}

Этот пример демонстрирует, что вам не обязательно называть каждую отдельную линию сетки, и что всегда остаются доступны номера линий, которые можно использовать помимо имён.

**Типовая разметка на гридах**

Давайте насладимся, как легко и красиво можно сделать обычно достаточно сложную разметку, используя номера линий:

HTML:

<div class="container">

<header class="header">header</header>

<aside class="sidebar">sidebar</aside>

<div class="content-1">Content-1</div>

<div class="content-2">Content-2</div>

<div class="content-3">Content-3</div>

<footer class="footer">footer</footer>

</div>

CSS:

.container {

display: grid;

width: 750px;

height: 600px;

grid-template-columns: 200px 1fr 1fr;

grid-template-rows: 80px 1fr 1fr 100px;

gap: 1rem;

}

.header {

grid-row: 1 / 2;

grid-column: 1 / 4;

}

.sidebar {

grid-row: 2 / 4;

grid-column: 1 / 2;

}

.content-1 {

grid-row: 2 / 3;

grid-column: 2 / 4;

}

.content-2 {

grid-row: 3 / 4;

grid-column: 2 / 3;

}

.content-3 {

grid-row: 3 / 4;

grid-column: 3 / 4;

}

.footer {

grid-row: 4 / 5;

grid-column: 1 / 4;

}

Или используя имена блоков:

.container {

display: grid;

width: 100%;

height: 600px;

grid-template-areas: "header header header"

"sidebar content-1 content-1"

"sidebar content-2 content-3"

"footer footer footer";

grid-template-columns: 200px 1fr 1fr;

grid-template-rows: 80px 1fr 1fr 100px;

gap: 1rem;

}

.header {

grid-area: header;

}

.sidebar {

grid-area: sidebar;

}

.content-1 {

grid-area: content-1;

}

.content-2 {

grid-area: content-2;

}

.content-3 {

grid-area: content-3;

}

.footer {

grid-area: footer;

}

Или с помощью именованных линий:

.container {

display: grid;

width: 100%;

height: 600px;

gap: 1rem;

grid-template-columns:

[main-start sidebar-start] 200px

[sidebar-end content-start] 1fr

[column3-start] 1fr

[content-end main-end];

grid-template-rows:

[row1-start] 80px

[row2-start] 1fr

[row3-start] 1fr

[row4-start] 100px

[row4-end];

}

.header {

grid-column: main-start / main-end;

grid-row: row1-start / row2-start;

}

.sidebar {

grid-column: sidebar-start / sidebar-end;

grid-row: row2-start / row4-start;

}

.content-1 {

grid-column: content-start / content-end;

grid-row: row2-start / row3-start;

}

.content-2 {

grid-column: content-start / column3-start;

grid-row: row3-start / row4-start;

}

.content-3 {

grid-column: column3-start / content-end;

grid-row: row3-start / row4-start;

}

.footer {

grid-column: main-start / main-end;

grid-row: row4-start / row4-end;

}

Все эти способы являются правильными.

**Гибкие гриды**

У грида есть две крайне мощные фичи для управления свободным местом. Эти фичи называются auto-fit и auto-fill, и используются внутри функции repeat, обычно с функцией **minmax**, как здесь:

grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

Они заменяют собой числа в repeat, и вычисляют максимально возможное число колонок в контейнере. Основное различие между ними – это то, как они справляются с лишним пространством в строке.

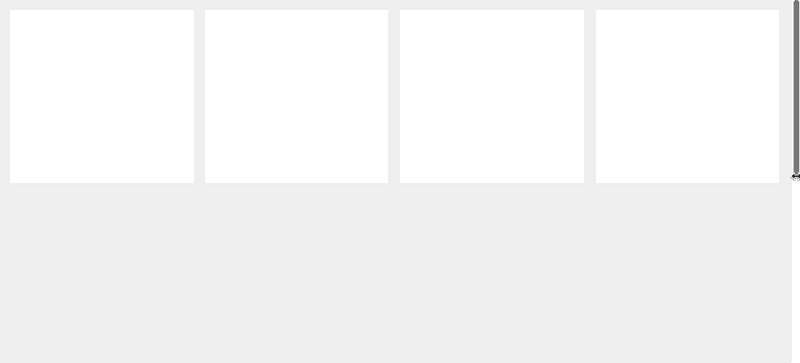
Auto-fill пытается поместить наибольшее количество повторяющихся элементов в колонку, сколько их может уместиться без переполнения. Когда места для добавления другого элемента не хватает, следующий элемент будет размещён на следующей строке, а это место, которое он не смог заполнить, останется пустым.



Ссылка на анимацию: <http://css-live.ru/Primer/grid1/1-6FrNqU9H5ZPGYGj8Bp347Q.gif>

Пример: После auto-fill могут оставаться пустые места, тогда как auto-fit будет схлопывать пустые места до 0px.

Auto-fit ведёт себя почти как \*\*auto-fill, за исключением того, что любая пустая область будет схлопываться и растягивать элементы в этой строке – напоминая поведение флексбоксов, когда по мере уменьшения доступного пространства колонки схлопываются.



Ссылка на анимацию: <http://css-live.ru/Primer/grid1/1-MPDRyNugrC0Xz-bhQ0e03g.gif>

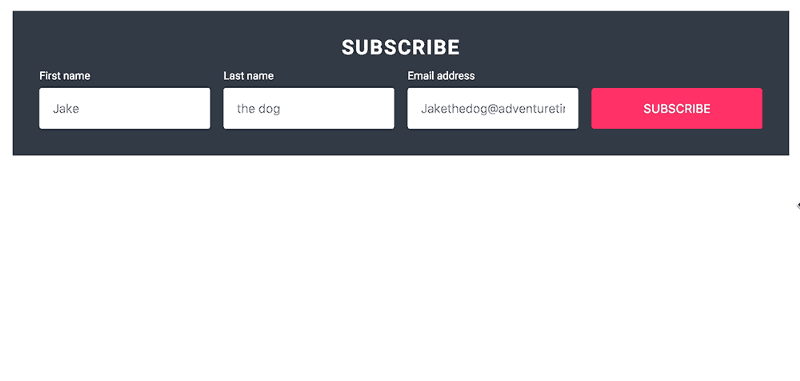
**Выводы**

Если бы меня спросили, что по моему мнению самое клевое в CSS Grid, то я бы ответила вот что: самым большим плюсом является то, насколько легко с его помощью создавать и поддерживать гибкие, адаптивные и красивые шаблоны без вреда хорошо структурированному, ровному, семантически чистому HTML.

**Домашнее задание на эту неделю:**

**Вопросы 💎**

1. Что за единица измерения - fr?
2. Как можно задать грид с 5 колонками шириной по 20%? Минимум 2 способа.
3. Самостоятельно разберитесь, что такое auto-fill и auto-fit ?
4. Как сделать такую табличку? Параметры: первая колонка шириной 100 пикселей, вторая 30%. Первая строчка высотой 200 пикселей, вторая строчка 100 пикселей.
5. Как сделать такое выравнивание в грид-контейнере?
6. Что такое и как задается *grid area*?
7. Приведите пример использования grid-template-areas (не копированием из этого урока 😉)
8. Каким свойством можно задать такое поведение элементов?



Ссылка на анимацию: <http://css-live.ru/Primer/grid1/1-MyO6enrNZEL9HSzYmdA8VA.gif>

1. Самостоятельно разберитесь, как работают именованные линии? Есть ли какие-то рекомендованные правила наименований? Если да, то какие?
2. Как проще всего задать 12 одинаковых по ширине колонок?

**Практическое задание 💎**

1. Пройти игру <https://cssgridgarden.com/> 🥰
2. Сверстать страничку с помощью CSS Grid, чтобы получить такой результат и прислать ссылку на github:
3. Сделать НОВУЮ страницу из макета Figma (найти в интернете, например, на [figmacrush](https://www.figmacrush.com/)) на базе технологии CSS Grid и прислать ссылку на github

Если вы хотите оставить свой репозиторий скрытым, то давайте доступ для аккаунта **itgirlschool**

ОТВЕТЫ

1. Это гибкая единица измерения, которая рассчитывается исходя из количества доступного пространства в grid-контейнере.
2. Grid-template-columns: 20% 20% 20% 20% 20%; grid-template-columns: repeat(5, 20%);
3. **Auto-fill** пытается поместить наибольшее количество повторяющихся элементов в колонку, сколько их может уместиться без переполнения. Когда места для добавления другого элемента не хватает, следующий элемент будет размещён на следующей строке, а это место, которое он не смог заполнить, останется пустым. **Auto-fit** ведёт себя почти как **auto-fill**, за исключением того, что любая пустая область будет схлопываться и растягивать элементы в этой строке – напоминая поведение флексбоксов, когда по мере уменьшения доступного пространства колонки схлопываются.
4. display: grid;

            grid-template-columns: 100px 30% 1fr;

            grid-template-rows: 200px 100px;

1. justify-content: space-between;
2. **grid-area** задает элементу имя, на которое можно сослаться при определении шаблона сетки, созданного с помощью свойства **grid-template-areas**. Является сокращенным свойством для следующих свойств:

-**grid-row-start** (определяет сколько строк будет занимать элемент, или на какой строке начинается элемент в макете сетки).

-**grid-row-end** (определяет сколько строк будет занимать элемент, или на какой строке завершится элемент в макете сетки).

-**grid-column-start** (определяет с какого столбца будет расположен элемент в макете сетки, или какое количество столбцов будет охватывать элемент).

-**grid-column-end** (определяет сколько столбцов будет занимать элемент, или на какой строке столбца завершится элемент)

7. .container {

    display: grid;

    grid-template-rows: 2% 1fr 2fr 10%;

    grid-template-columns: 1fr 1fr;

    grid-template-areas:

        "header header"

        "main aside"

        "services services"

        "footer footer";

}

Пример из моей работы (п.№3 из практической части дз)

8. auto-fit

9.

10. grid-template-columns: repeat(12, размер колонки);