Адаптивная верстка. Основы CSS.

CSS (Cascading Style Sheets) – это язык иерархических правил (таблиц стилей), используемый для представления внешнего вида документа, написанного на HTML или XML. Слово «cascading – каскадный» здесь означает то, что стиль, применённый к родительскому элементу, будет применяться и ко всем дочерним элементам внутри него.

Стили могут быть применены к HTML-документу следующими способами:

- Inline – путём задания стилей в атрибуте style непосредственно на целевом элементе;
- Internal – то есть задание стилей в тэге <style>, расположенном в тэге-секции <head> документа;
- External – использование тэга <link rel="stylesheet" href="*.css" /> в секции <head> для включения внешнего *.css файла со стилями.

В крупных проектах, как правило, используется последний вариант. Подробнее можно узнать в [1]. Для тех кто не знаком с CSS – краткое введение в [2].

Синтаксис правил CSS в общем виде представляет собой набор блоков следующего вида (здесь и ниже квадратные скобки для визуального отделения и в синтаксисе не присутствуют):

[selectors group] [declaration block]

где [selectors group] – это один селектор или несколько селекторов, отделённые друг от друга запятой:

[selector_1],
[selector_2],
…

а [declaration block] – набор деклараций, отделённых друг от друга точкой с запятой и расположенных совместно в блоке из фигурных скобок:

```
{
        [declaration_1];
        [declaration_2];

        [declaration_3];
        …
}
```

В свою очередь, декларация [declaration] имеет синтаксис:

        [property]: [value]

Селектор – это директива выбора элемента в документе. Он может быть простым указателем на тэг (например, div), может быть указателем на класс (записывается с точкой перед названием класса, например, .image), может быть указателем на id (записывается с символом решётки перед id, например, #container), может быть указателем на атрибут (записывается в скобках, например, [checked]), а также может быть универсальным селектором (записывается как *), применимым ко всем элементам документа.

Если Вы плохо разбираетесь в селекторах, рекомендуем поиграть в этот тренажёр (его страница прогружается долго, нужно подождать около минуты для подгрузки списка селекторов в левой панели) [3].

Свойство (property) – это один из многих идентификаторов, указывающих, что именно нужно установить. Например, длина или высота. Значение (value) – конкретная величина или значение для данного свойства.

На скриншоте представлен пример одного из правил CSS:

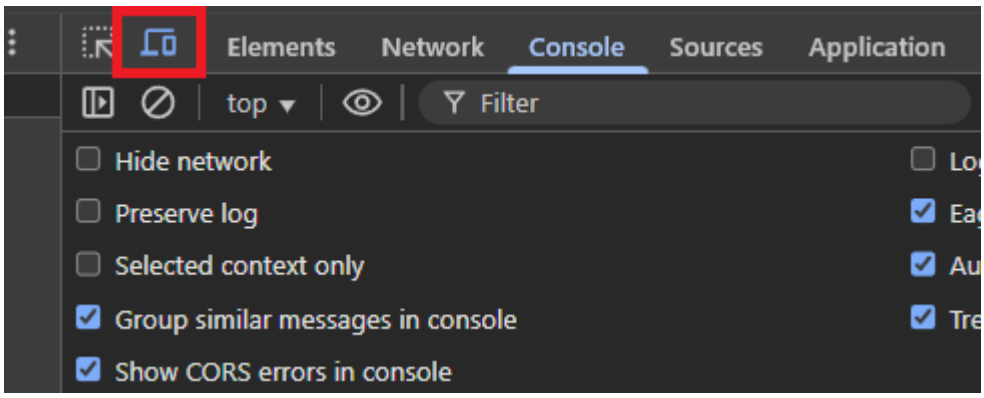
Формальный синтаксис правил CSS, а также список практически всех свойств представлен в [4]. Небольшая шпаргалка по селекторам представлена в [5], а по комбинаторам – в [6].

CSS хорошо осваивается на практике. Поэтому дальнейшее изучение будет происходит в процессе создания реального проекта, которое ожидает Вас в последующих занятиях. А сейчас разберём понятия адаптивной и отзывчивой верстки.

Отзывчивая верстка (responsive layout) – это верстка, которая плавно адаптирует контент по ширине и высоте вьюпорта (viewport), принципиально не изменяя расположение элементов в зависимости от хоста (мобильное устройство, экран планшета или экран монитора) или назначения.

Отзывчивая верстка является частным случаем адаптивной верстки (adaptive layout) – это верстка, которая адаптирована под различные разрешения вьюпорта, что, как правило, определяется хостом. В этом случае расположение и наличие элементов верстки может принципиально отличаться в зависимости от разрешения и размера вьюпорта.

Например, сайт Авито для десктопа (https://avito.ru/) имеет дизайн, принципиально отличающийся от мобильной версии (https://m.avito.ru/). Можете проверить это самостоятельно, воспользовавшись инструментами разработчика браузера (DevTools) для переключения расширений вьюпорта:



В действительности, практически нет web-приложений или сайтов, которые являются чисто отзывчивыми. Например, Вовлекай [7] использует комбинацию отзывчивой верстки и адаптацию через брейкпоинты.

Брейкпоинты (breakpoints) – это заданные точки, привязанные к определённой ширине или высоте вьюпорта, проходя через которые, верстка изменяется принципиально. Как правило, брейкпоинты задают

в CSS. Например, в проекте Вовлекай [7] мы используем следующие брейкпоинты для ширины вьюпорта (единицы указаны в пикселях):

```
export const BREAKPOINTS = {
    xs: 0,
    sm: 375,
    md: 768,
    lg: 1200,
    xl: 1536
};

export default BREAKPOINTS;
```

Попробуйте проверить это самостоятельно, изменяя ширину экрана через инструменты разработчика в браузере.

Для задания брейкпоинтов используются медиа-запросы (media-queries) CSS.

Медиа-запрос – это условие или состояние устройства, доступное браузеру, при выполнении заданного значения которого будут применены соответствующие правила CSS. Записывается медиа-запрос через CSS-директиву @media. Формальный синтаксис имеет следующий вид:

```
@media media-type and (media-feature-rule) {
  /* CSS rules go here */
}
```

где media-type – это код самого устройства, media-feature-rule – это конкретное значение состояния устройства. Под устройством имеется в виду экран (screen), принтер (print) или всё (all). Для дальнейшего выполнения заданий Вам необходимо изучить медиа-запросы по статье [8] или [9].

Для создания адаптивной верстки применяются также следующие две очень часто используемые технологии – Flexbox и Grid Layout. Вы также будете применять их бесчисленное множество раз в своих проектах.

Flexbox – сокращение от Flexible Box Module – это технология верстки с использованием flex-контейнеров. Flex-контейнером может быть любой HTML-элемент, если задать ему CSS-свойство display в значение flex. Такой элемент определяет правила выравнивания дочерних элементов контейнера по одной из осей (называемой главной) – вертикальной или горизонтальной (по умолчанию). В статье [11] перечислен тот минимум свойств, которые необходимо знать и понимать для дальнейшей работы – это flex-direction, gap, justify-content, align-items, а также чуть реже используемые flex-shrink, flex-grow и flex-basis. Также рекомендуется посмотреть видео [12] и пройти небольшое обучение в [14].

Также отдельно отметим, что центрировать некий элемент по вертикальной оси относительно родительского элемента можно только с помощью конвертации родительского элемента во flex-контейнер и установки свойства align-items: center (если главная ось установлена горизонтально), либо justify-content: center (если главная ось установлена вертикально). В то время как центровка по горизонтальной оси может быть осуществлена как с помощью flex, так и через margin: 0 auto (или margin-left: auto, margin-right: auto).

Grid Layout – это технология верстки с использованием grid-контейнеров. Чтобы сделать элемент grid-контейнером, необходимо задать ему CSS-свойство display в значение grid. В отличие от Flexbox, здесь для выравнивания дочерних элементов контейнера используется сразу две оси – вертикальная и горизонтальная.

В Grid Layout добавляются такие свойства, как grid-template-columns (задание кол-ва и размерности столбцов), grid-template-rows (задание кол-ва и размерности строк), grid-template-areas (задание именованных областей по нескольким столбцам и строкам одновременно) и другие. Обязательно разберитесь с этими свойствами, так как они понадобятся в дальнейшей работе. Про Grid Layout также описано в [11], видео [13]. Небольшое обучение, как и с Flexbox, в [15].

Ещё одна тема, которую стоит упомянуть, хотя она и не относится к адаптивной верстке – это свойство z-index и контекст наложения (stacking context).

HTML-элементы располагаются не только в плоскости экрана, но также могут быть наложены друг на друга, если они спозиционированы

через position: relative или иным способом (например, через Grid Layout). То есть, располагаются вдоль оси z, перпендикулярной плоскости экрана. Те элементы, что были монтированы в DOM-дерево позже других, при прочих равных условиях будут располагаться выше. Однако можно изменить это поведение, если задать CSS-свойству z-index определённое числовое значение. Те элементы, у которых z-index выше, будут располагаться над теми, у кого z-index не задан или числовое значение ниже. Однако это правило не распространяется на весь документ – это происходит в контексте наложения (stacking context).

Контекст наложения – это HTML-элемент, удовлетворяющий определённым условиям, которые возникают при определённой комбинации CSS-свойств и их значений. Например, элемент может стать контекстом наложения, если прописать ему следующие стили: position: relative; z-index: 0.

Дочерние элементы контекста наложения подчиняются стандартному позиционированию (то есть по моменту монтирования в DOM) внутри него и никак не связаны с дочерними элементами другого контекста наложения и с самим этим контекстом наложения. При этом, два контекста наложения также починяются стандартному позиционированию, если конечно, они не находятся в общем контексте наложения.

Если Вы запутались, то есть хорошая статья с картинками (были и получше, но за 3 года потерялись), на Доке [16]. Обратите внимание на все перечисленные условия для создания из элемента контекста наложения. Запоминать их не нужно – всегда можно заглянуть в справочник, например, на MDN.

Вопросы и задания

1. Чем отличаются друг от друга элементы с CSS-свойствами display в значениях «block», «inline» и «inline-block»? Какие ещё значения свойства display Вы знаете? Перечислите их и кратко укажите, для чего используется каждое из них (таблицы можно пропустить – про них вообще можно забыть).

   Если затрудняетесь – вот хорошая статья в помощь [17].

2. Для чего используется CSS-свойство position? Перечислите его возможные значения и их назначение.

3. Дан массив данных следующего вида:

```
const badges = [
    { id: 1, url: “#55a3c8” },
    { id: 2, url: “#ffa500” },
    { id: 3, url: “#10461c” },
    { id: 4, url: “#008000” },
    { id: 5, url: “#8d8d5f” },
    { id: 6, url: “#72de25” },
    { id: 7, url: “#96825d” },
].
```

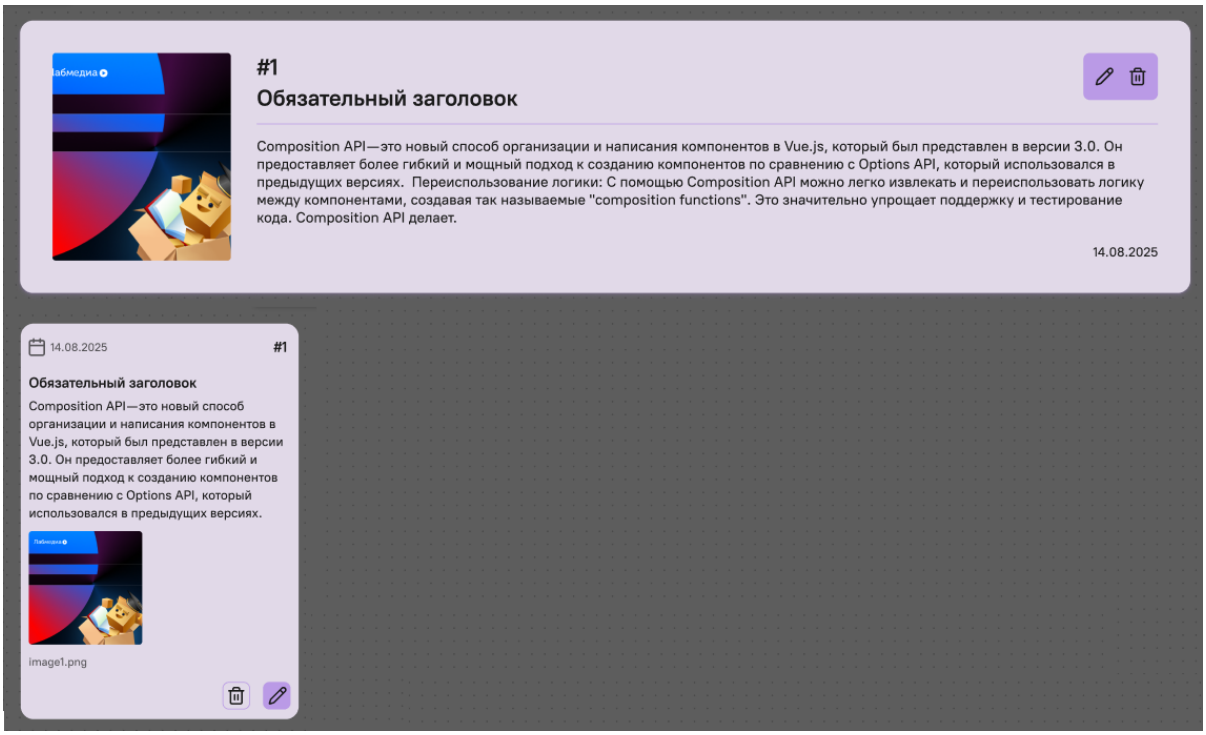
   Создайте HTML-страницу, отображающую список круглых жетонов 50 на 50 пикселей с границей 2 пикселя в следующем виде без использования z-index и технологии Grid Layout:

   где выводятся первые 5 жетонов из массива badges, причём первый жетон находится в крайней правой позиции (то есть выводятся с права налево). Цвета жетонов соответствуют полю url. Используйте классы для обращения к элементам в правилах CSS.

4. Дополните предыдущее задание, добавив брейкпоинт на 375px. Если ширина вьюпорта меньше или равна брейкпоинту, то должны отображаться только 3-ий, 4-ый и 5-ый жетоны в том же порядке.

   Также дополните объекты массива badges полем image и присвойте каждому жетону в поле image ссылку на изображение из сети Интернет, либо воспользуйтесь предварительно загруженными изображениями из папки с проектом. Изображение должно быть отцентрировано и растянуто на весь жетон, чтобы не было пустых областей. В качестве референса можно взять достижения из Вовлекай [7]:

5. Сверстайте следующую карточку согласно макету из фигмы [10]. Первый вариант соответствует ширине вьюпорта, меньшей или равной 375px, а второй вариант – остальные величины ширины. Сможете ли Вы сделать это без дублирования элементов карточки? Воспользуйтесь изученными в этой теме возможностями и технологиями CSS.



Желательно, чтобы элемент даты показывал текущую дату при каждом запуске HTML-файла.

Ссылки

[1]. https://www.w3schools.com/html/html_css.asp

[2]. https://developer.mozilla.org/ru/docs/Learn_web_development/Core/Styling_basics/Getting_started

[3]. https://www.w3schools.com/cssref/trysel.php

[4]. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

[5]. https://www.w3schools.com/cssref/css_selectors.php

[6]. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Combinators

[7]. https://labmedia.vovlekay.online/

[8]. https://doka.guide/css/media/

[9]. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Media_queries

[10]. https://www.figma.com/board/UK7yR8nqWpGJlkJGl28chk/%D0%A8%D0%BA%D0%BE%D0%BB%D0%B0_frontend_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA%D0%B0?node-id=0-1&p=f&t=bOSL2DflJq0mNLAH-0

[11]. https://habr.com/ru/companies/ruvds/articles/523808/

[12]. https://www.youtube.com/watch?v=YG8Vhz1pAsU

[13]. https://www.youtube.com/watch?v=JrKOHNRnRMg

[14]. https://www.w3schools.com/css/css3_flexbox.asp

[15]. https://www.w3schools.com/css/css_grid.asp

[16]. https://doka.guide/css/stacking-context/

[17]. https://learn.javascript.ru/display