Приднестровский государственный университет им. Т.Г. Шевченко
Инженерно-технический институт

РАЗРАБОТКА АДАПТИВНОЙ ВЁРСТКИ ВЕБ-САЙТА ПРИ ПОМОЩИ ТЕХНОЛОГИЙ HTML5 И CSS3

Лабораторный практикум

Разработал:
ст. преподаватель
кафедры ИТиАУПП
Бричаг Д.В.

г. Тирасполь
2021 г.

Лабораторная работа №7

Разработка UI веб-сайта

Цель работы: ознакомиться с принципами контроля версий. Ознакомиться с хранилищами репозиториев. Научиться пользоваться командами git и хранить кодовую базу в удалённых репозиториях. Публиковать проекты в глобальной сети при помощи автоматических сервисов. Научиться разрабатывать вёрстку при помощи новых версий CSS-фреймворков.

Теоретическая справка

Сетки Grid в современном браузере

Сетки (grids) являются установленным инструментом проектирования и многие современные макеты веб-сайта основаны на регулярной сетке. В этой статье мы рассмотрим дизайн на основе сетки и увидим как CSS можно использовать для создания сеток — как с помощью современных инструментов, так и с помощью новых технологий, которые только начинают становиться доступными в браузерах.

W3C описывает модуль CSS Grid Layout как систему двумерного макета, оптимизированного для дизайна пользовательского интерфейса. Главная идея, лежащая в основе макета сетки, заключается в разделении веб-страницы на столбцы и строки. В образовавшиеся области сетки можно помещать элементы сетки, а управлять их размерами и расположением можно с помощью специальных свойств модуля.

Кроме того, благодаря своей способности явно размещать элементы в сетке, Grid Layout позволяет кардинально преобразовывать структуру визуального макета (отображаемого на экране), не требуя соответствующих изменений разметки.

Хотя многие макеты могут быть отображены с помощью Grid или Flexbox, у каждого есть свои особенности. Grid обеспечивает двухмерное выравнивание, использует нисходящий подход к макету, допускает явное перекрытие элементов и обладает более мощными связующими возможностями. Flexbox фокусируется

на распределении пространства по оси, использует более простой восходящий подход к макету, может использовать систему переноса строк на основе размера контента для управления своей вторичной осью и опирается на базовую иерархию разметки для построения более сложных макетов.

Grid модуль для CSS был разработан рабочей группой CSS для того, чтобы сделать создание шаблонов в CSS максимально удобным. Он попал в рекомендации по официальному внедрению в феврале 2017 года, а основные бразуеры начали его поддержку уже в марте 2017 года.

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

С этим мощным функционалом и интуитивно понятным синтаксисом, шаблоны на grid несомненно будут менять представление о создании веба как такового.

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. Обычно первым шагом является определение того, сколько колонок и рядов есть в гриде. Но даже это опциональный момент — как вы увидите далее.

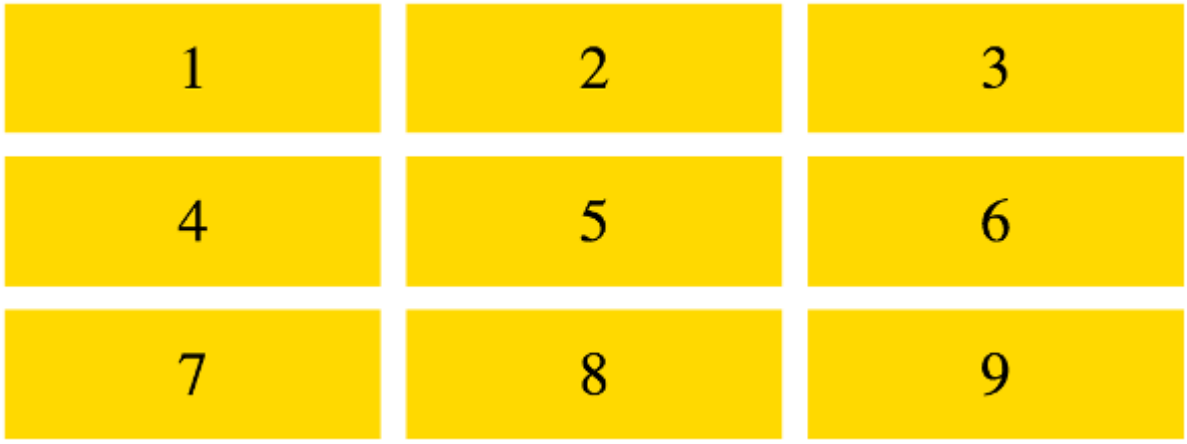
Это пример грида с четырьмя рядами и тремя колонками. Он состоит из 12 grid элементов. Каждый из этих элементов отмечен зеленым и между ними есть небольшое расстояние.

Все эти grid элементы одного размера, но они могли бы быть совершенно любого размера, такого, какого мы захотим. Мы могли бы сделать их совершенно разными по размерам, если бы захотели. Некоторые из них могли бы охватывать несколько столбцов и рядов, другие могли бы оставаться размеров с одну ячейку.

Далее в этой статье вы узнаете об этих и других функциях в grid, некоторые из которых вполне смогут вас удивить.

Создаем Grid

Вот пример простого 3x3 грида с небольшими отступами между элементами.



А вот код:

```
<!doctype html>
<title>Example</title>
<style>
#grid {
  display: grid;
  grid-template-rows: 1fr 1fr 1fr;
  grid-template-columns: 1fr 1fr 1fr;
  grid-gap: 2vw;
  }
#grid > div {
  font-size: 5vw;
  padding: .5em;
  background: gold;
  text-align: center;
```

```
}
</style>
<div id="grid">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
</div>
```

Давайте внимательнее вглядимся в код. HTML разметка для CSS Grid выглядит вот так:

```
<div id="grid">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
</div>
```

Таким образом мы видим обычный HTML, состоящий из элементов, вложенных в свой внешний элемент. Но именно для наших целей, внешний

это контейнер гридов. Соотвественно, все элементы вложенные в него будут являться грид элементами.

Но по факту, это не будет полноценными гридом, пока мы не применим кое-какой CSS для него. Вот код, который создаёт его:

```
#grid {
  display: grid;
  grid-template-rows: 1fr 1fr 1fr;
  grid-template-columns: 1fr 1fr 1fr;
  grid-gap: 2vw;
}
```

Это правило применяется к внешнему , так как ему было назначено ID #grid.

Вот объяснение того, что написано в этом CSS:

display: grid

Превращает элемент в grid контейнер. Это все, что нужно для того, что создать грид. Теперь у нас есть грид-контейнер и грид-элементы. Значения гридов создают блочный контейнер. Вы так же можете использовать display: inline-grid, что создать строчный грид-контейнер. Или же вы можете использовать display: subgrid, чтобы создать подсетку, это значение используется на самих grid элементах.

grid-template-rows: 1fr 1fr 1fr

Выстраивает ряды в гриде. Каждое значение представляет размер ряда. В этом случае все значения равны 1fr. Очень подробно и понятно про (fr) можно почитать тут.

Но конечно же, для этого можно было бы использовать разные значения, такие как 100px, 7em, 30% и так далее. Вы также можете назначать имена строкам вместе с их размерами.

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

Тоже самое, что и выше, только определяет колонки в гридах.

grid-gap: 2vw

Выставляет разрыв. То есть пробелы между грид элементами. Тут мы используем vw единицу, которая относительна ширине viewport, но также мы можем использовать 10px, 1em и т. д. Grid-gap свойство это сокращение для grid-row-gap и grid-column-gap свойств.

Ну, а другая часть кода просто назначает разные стили грид элементам.

```
#grid > div {
  font-size: 5vw;
  padding: .5em;
  background: gold;
  text-align: center;
}
```

Функция repeat()

Вы можете использовать функцию repeat() для повторяющихся объявлений значения размера элемента. Для примера, вместо того, чтобы делать это:

```
grid-template-rows: 1fr 1fr 1fr 1fr 1fr;
```

Мы можем сделать так:

```
grid-template-rows: repeat(5, 1fr);
```

Что значительно сократит количество кода, которое вам нужно написать, особенно, если вы работаете с большими и часто повторяющимися элементами в гридах.

Исключение элементов из выборки

Осторожно применяя :not(), можно исключить элементы из выборки.

:not() прекрасно подходит для использования во фреймворках или системах проектирования для повышения специфичности классов. Речь идет о классах, которые потенциально могут примениться к чему угодно и относительно которых известно, что в определенных комбинациях они могут быть источниками проблем.

Рассмотрим расширенный пример исключения ссылок с классами. Допустим, вы регулируете контраст для текста (возможно, в контексте темного режима) и хотите применить настройку контрастности также к текстовым ссылкам:

```
/* Светлая тема страницы */
a:not([class]) {
  color: blue;
}
/* Цвет текста для тёмной темы */
.dark-mode {
  color: #fff;
}
/* Наследование цвета ссылок через свойство inherit */
.dark-mode a:not([class]) {
  color: inherit;
}
```

Из селекторов :not() можно даже составить цепочку. Скажем, вы хотите создать правило для полей ввода в форме, но оно не должно касаться определенных типов:

```
input:not([type="hidden"]):not([type="radio"]):not([type="checkbox"])
```

Внутрь :not() можно поместить другой псевдоселектор. Например, чтобы исключить состояние :disabled для кнопок.

```
button:not(:disabled)
```

Таким образом вы сначала сбросите стили кнопки, и только затем примените цветовые стили, границы и пр. к неотключенным кнопкам — вместо последующего удаления этих стилей для button:disabled. Это позволит вам написать более аккуратные правила.

Эффективная выборка групп элементов

Псевдокласс :is(), недавно получивший поддержку, «принимает список селекторов как аргумент, и выбирает любой элемент, который может быть выбран одним из селекторов в этом списке» (документация MDN).

Этот псевдокласс позволяет более компактно выбирать элементы типографии:

:is(h1, h2, h3, h4)

Или более лаконично охватить стили макета:

:is(header, main, footer)

Мы даже можем скомбинировать :is() с :not() и выбрать, например, элементы, не являющиеся заголовками:

:not(:is(h1,h2,h3,h4))

Если вы хотите начать пользоваться этим селектором, пока нужно будет включать по крайней мере webkit-префикс версию. Из-за того, что браузеры порой весьма странно используют селекторы, вам нужно будет делать :is() уникальным правилом, отделенным от is(), чтобы браузер не выбросил оба правила.

:-webkit-any(header, main, footer)

Практическая часть

Ещё одним распространённым видом отображения страниц в сети является страница с боковой панелью. В таком виде можно встретить страницы от простых личных блогов, до крупнейших сайтов СМИ и социальных сетей. Вид такой страницы настолько популярен, что для боковой панели в стандарт HTML5 был введён отдельный тег <aside> – который можно так и перевести: в стороне, с боку.

Создадим такую страницу, которая будет выполнять функцию блога. Такие страницы встречаются у крупных магазинов или сервисов, где публикуются новости компании, товаров или иных событий в жизни компании.

Аналогично лабораторной работы №6 создадим страницу blog.html и добавим на неё ссылки в меню.

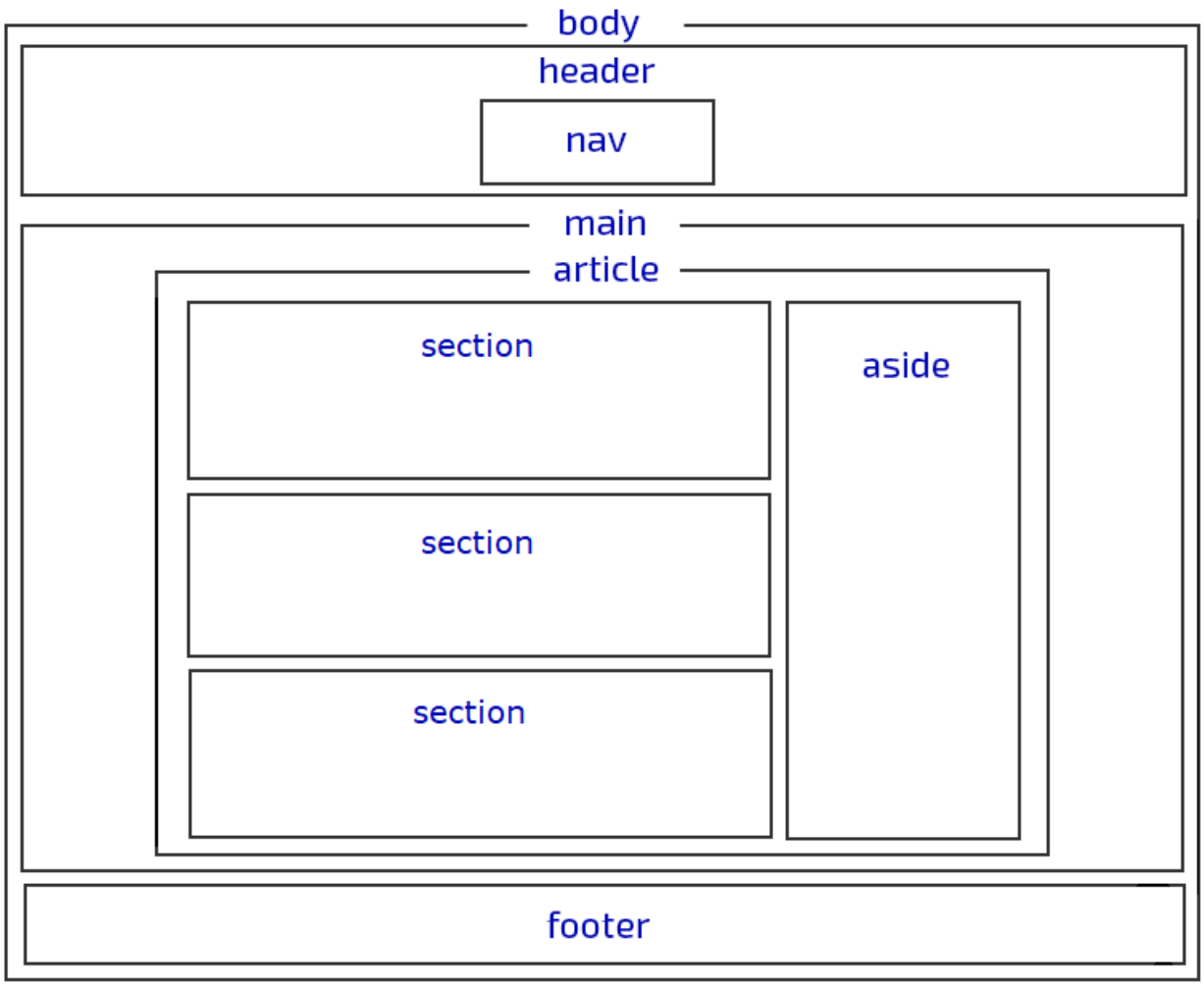


Рис. 118 – Общий вид разметки страницы

В сам html документ скопируем основные элементы из страницы index.html: основное тело документа, шапку, секцию с обложкой и подвал. Но внутри main после обложки уже не будут идти обычные секции по очереди. Они будут частью основного блока article, где слева будут расположены секции, а справа боковая панель aside. Схематическое изображение страницы с основными блоками – тегами представлено на рисунке 118. А на рисунке 119 html код такой страницы.

```
19  <body>
20    <header>
21      <nav id="myTopnav">
22        <img src="img/fort-icon.png" alt="Логотип сайта">
23        <ul>
30        <div id="hamburger" class="icon">
31          &#9776;
32        </div>
33      </nav>
34    </header>
35    <main>
36      <section id="top">
37        <img class="top-image top-image-sub" src="img/02.jpg" alt="Обложка стран
38      </section>
39      <article class="blog-article">
40        <div class="blog-container">
41          <!-- Запись в блоге -->
42          <section class="blog-card">
72        </div>
73        <aside class="blog-aside">
74
75        </aside>
76      </article>
77
78    </main>
79    <footer>
80      <nav>
99      <div class="brand">
103     <div class="social">
120     <div class="copyright">© Very First Page 2021</div>
121   </footer>
122 </body>
```

Рис. 119 – Общий код страницы блога

Здесь, чтобы было легче работать со стилями мы можем создать ещё один файл стилей, например, blog.css и добавить его в head страницы. В сам файл добавим основные стили крупных блоков и подготовим медиа-запросы для адаптивности страницы.

```css
.blog-article {
  display: flex;
  max-width: 960px;
  margin: 70px auto 30px;
}
.blog-container {
  flex: 2;
  padding: 30px 15px;
}
.blog-aside {
  flex: 1;
  padding: 30px 15px;
}

@media screen and (max-width: 992px) {

}

@media screen and (max-width: 768px) {

}

@media screen and (max-width: 576px) {

}
```

Рис. 120 – Стили основных блоков страницы

На самой странице пока ничего нет. Наша секция с классом blog ещё пустая, и на боковую панель ничего не добавлено. Начнём с добавления записей блога. Как вы могли заметить выше, каждая запись в блоге будет создаваться как секция с классом “blog-card”. Такие записи будут следовать друг за другом внутри блока div.blog-container. Заполним одну такую запись на странице нашего блога:

1. Мысленно разделим её на условные три части: верхнюю часть, тело записи и подвал.

2. В верхней части мы создадим обложку статьи, тут же можно добавить автора поста, раздел блога, количество комментариев и другую информацию.

3. В теле записи будет находиться заголовок статьи, сама статья или её краткая запись и теги.

4. В подвале – дата записи. Тут же можно добавить социальные кнопки или функции, как например лайки или кнопки быстрых действий.

Сам HTML код будет выглядеть примерно, как на рисунке 121. Вы же по желанию можете в него добавить другие опции, о которых упоминалось выше.

```
<div class="blog-container">
  <!-- Запись в блоге -->
  <section class="blog-card">
    <div class="blog-header">
      <div class="blog-cover">
      </div>
    </div>
    <div class="blog-body">
      <div class="blog-title">
        <h2>Lorem ipsum dolor sit amet, consectetur adipisicing elit</h2>
      </div>
      <div class="blog-text">
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed eiusm
          Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris u
          in reprehenderit in voluptate velit esse cillum dolore eu fugiat par
           sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
      </div>
      <div class="blog-tags">
        <ul>
          <li><a href="#">lorem</a></li>
          <li><a href="#">proident</a></li>
          <li><a href="#">amet</a></li>
          <li><a href="#">laborum</a></li>
        </ul>
      </div>
      <div class="blog-footer">
        <div class="blog-published-date">
          3 дня назад
        </div>
      </div>
    </div>
  </section>
</div>
```

Рис. 121 – Вёрстка и наполнение записи блога

И для этой вёрстки нам потребуется несколько стилей. Для моего блога я прописал стили, как на рисунках 122 и 123.

```
.blog-card {
  background: #f7f7f7;
  margin: 0 auto 50px;
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);
  border-radius: 12px;
  overflow: hidden;
}
.blog-cover {
  background: lightgray url('../img/blog/blog02.jpg') center no-repeat;
  background-size: cover;
  height: 200px;
}
.blog-body {
    padding: 27px;
}
.blog-title {
  margin-bottom: 25px;
}
.blog-title h2 {
  font-weight: 100;
  font-size: 30px;
  line-height: 1.3em;
}
.blog-title {
  line-height: 1.5em;
  color: #464646;
}
```

Рис. 122 – Стили для контейнера, заголовка и обложки

Для тегов и подвала стилей чуть больше, чтобы добиться типового внешнего вида записи с активными элементами. В нашем случае это полоса тегов и информация о дате публикации.

```css
.blog-tags ul {
  display: flex;
  justify-content: flex-start;
  list-style: none;
  background: lightgray;
  padding: 10px;
  border-radius: 5px;
  margin: 25px 0;
}
.blog-tags li {
  padding: 0 5px 0 0;
}
.blog-tags a {
  text-decoration: none;
  color: #464646;
  border: 1px solid gray;
  border-radius: 5px;
  padding: 3px 10px;
  font-size: 14px;
  background: white;
  transition: all 250ms ease;
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0);
}
.blog-tags a:hover {
  border: 1px solid #464646;
  box-shadow: 2px 2px 4px rgba(0, 0, 0, 0.2);
}
.blog-footer {
  border-top: 1px solid lightgray;
  padding: 12px 0;
  display: flex;
  justify-content: flex-end;
  text-transform: uppercase;
  font-size: 14px;
  font-weight: 400;
  color: gray;
}
```

Рис. 123 – Стили для тегов и подвала

По итогу получилась такая аккуратная карточка, как на рисунке 124. Постарайтесь поэкспериментировать со стилями и создать свой уникальный дизайн. У вас точно получится интересно.

Рис. 124 – Запись блога

Теперь если добавить ещё одну копию записи, то она поместится снизу, а между записями останется место. Таким образом можно будет получать записи от сервера и выводить их на странице по очереди, и не переживать о том, что вёрстка сломается.

Вы можете создать иной порядок отображения записей, например, в две колонки аналогично тому, как в предыдущих работах размещались карточки с изображениями или текстом.

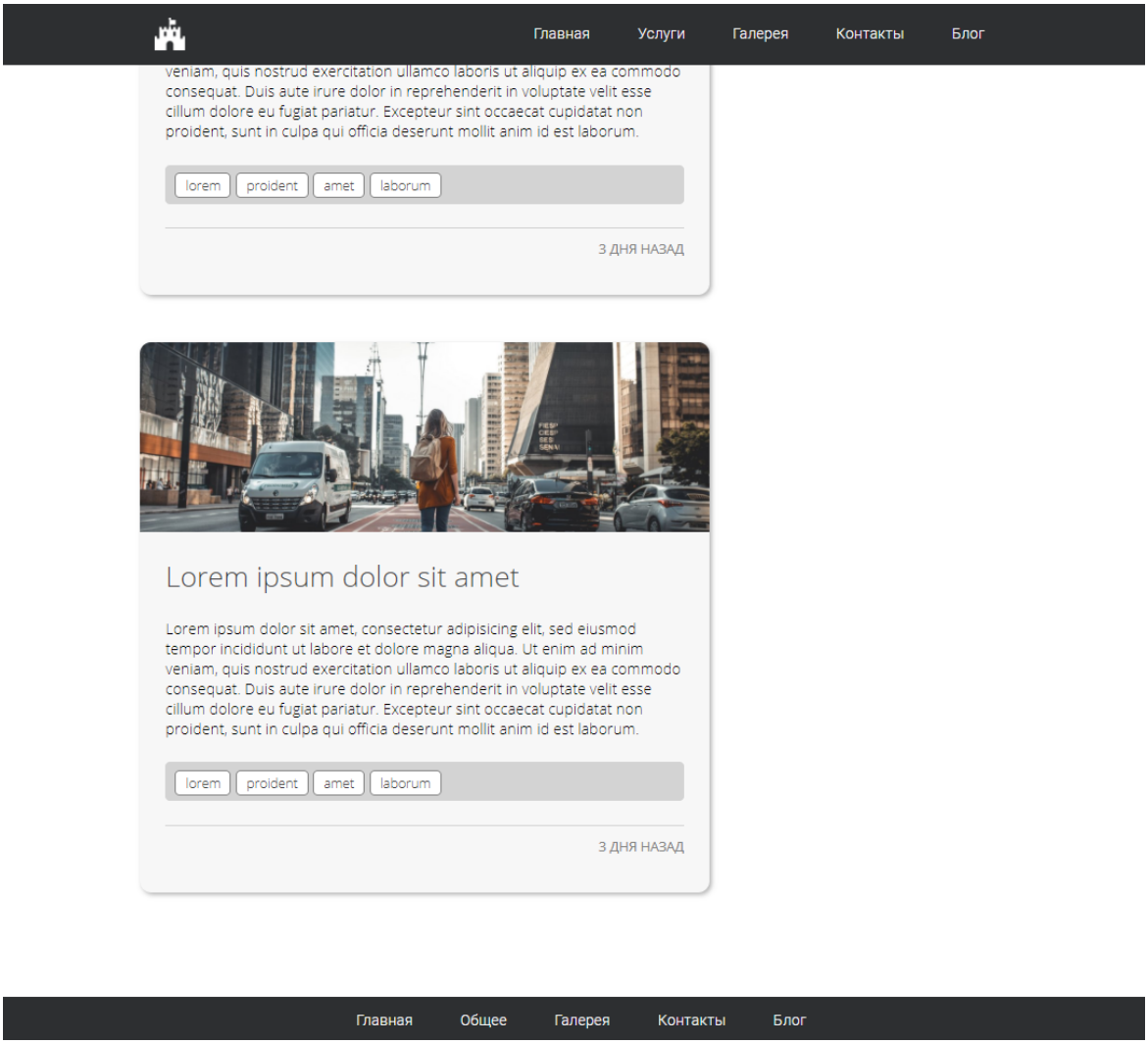


Рис. 125 – Ещё одна запись блога

Как вы могли заметить, карточки смещены влево от центра страницы. Всё потому, что мы в начале задали стили в том числе и для боковой части. Давайте заполним и её – создадим два блока: с поиском и популярными записями.

```
103          <aside class="blog-aside">
104            <div class="aside-search">
105
106            </div>
107            <div class="aside-recent">
108
109            </div>
110          </aside>
111        </article>
```

Рис. 126 – Два блока внутри тега aside

Так как они будут расположены в одной секции и выполнять схожие действия с точки зрения дизайна, то их стили будут одинаковыми.

```
.aside-search,
.aside-recent {
  background: #f7f7f7;
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);
  border-radius: 12px;
  padding: 45px 25px 50px;
  margin-bottom: 50px;
}
```

Рис. 127 – Стили для медиа-запроса 576px

Схожие стили мы использовали для карточки записи блога. Таким образом мы создаём единообразие внешнего вида страницы.

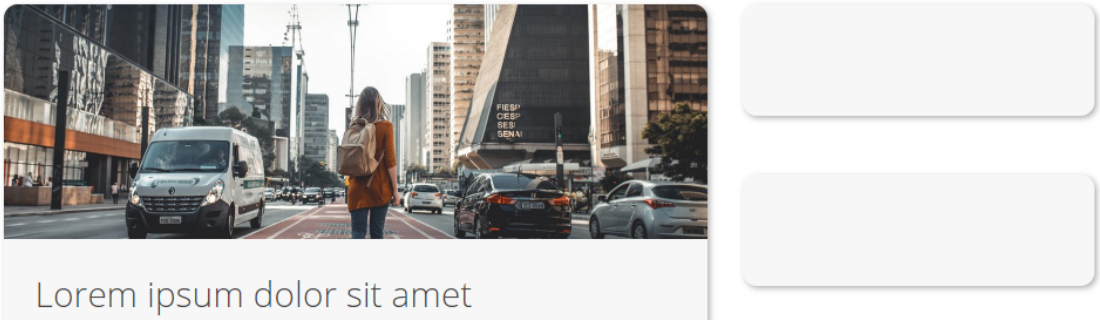


Рис. 128 – Блоки на боковой панели

Добавим в верхний блок заголовок, разделитель и два поля: типа поиск и подтвердить.

```
<div class="aside-search">
  <h3 class="search-title">Поиск в блоге</h3>
  <div class="search-delimiter"></div>
  <input class="search-text" type="search" name="search-text" value="" placeholder="Введите слово"
  <input class="search-do" type="submit" name="search-do" value="Искать">
</div>
```

Рис. 129 – Разметка блока с поиском

И пропишем им стили. Здесь также одинаковые стили применяем к двум классам, прописав их через запятую.

```
.search-title {
  font-size: 25px;
  color: #454546;
}
.search-delimiter {
  width: 100px;
  margin: 25px 0 37px;
  height: 1px;
  background: rgba(206, 65, 110, 0.74);
}
.search-text, .search-do {
  border: 1px solid #ce416e;
  padding: 15px;
  border-radius: 6px;
}
.search-text {
  width: 170px;
}
.search-do {
  background: #ce416e;
  color: white;
  cursor: pointer;
}
```

Рис. 130 – Стили для элементов поиска

А те стили, что уникальны, прописываем для каждого класса по отдельности.

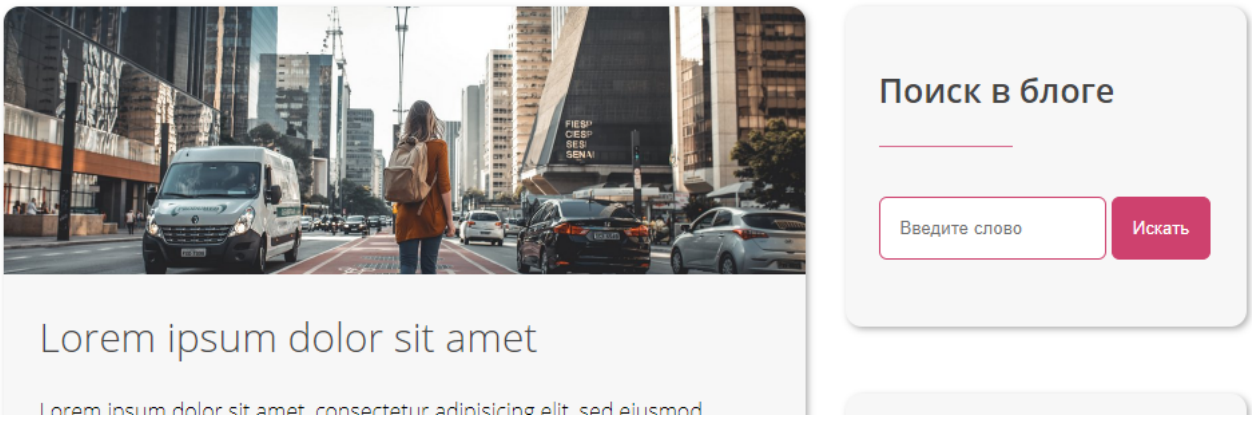


Рис. 131 – Внешний вид блока поиска на странице Блог

Для второго блока продублируем заголовок и разделитель с классами search-title и search-delimeter. Это другой подход к переиспользованию стилей, когда мы не назначаем в файл стилей новый класс, а применяем его к элементам из другого блока в html.

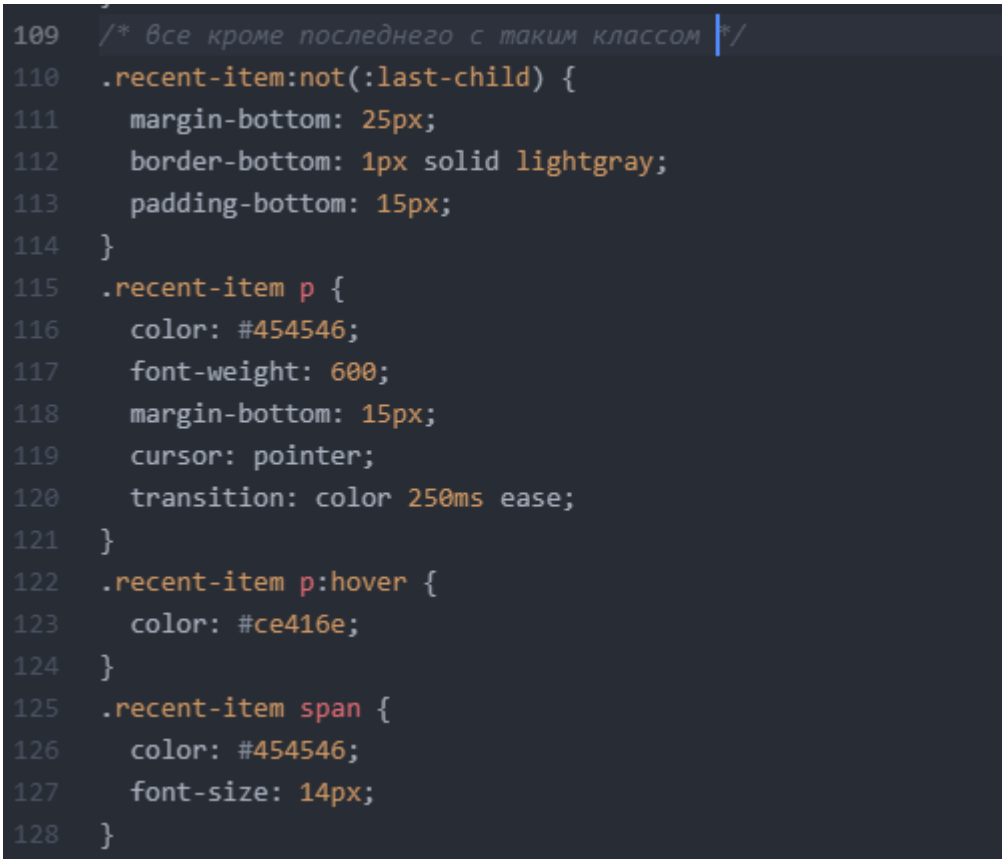
```
<div class="aside-recent">
  <h3 class="search-title">Популярные статьи</h3>
  <div class="search-delimiter"></div>
  <div class="recent-item">
    <p>Ut enim ad minim veniam, quis nostrud</p>
    <span>15/10/2021</span>
  </div>

  <div class="recent-item">
    <p>Excepteur sint occaecat cupidatat</p>
    <span>08/09/2021</span>
  </div>

  <div class="recent-item">
    <p>Duis aute irure dolor in reprehenderit</p>
    <span>29/08/2021</span>
  </div>
</div>
```

Рис. 132 – Разметка блока «популярных статей» блога

Для остального текста назначаем новые стили. Обратите внимание на первый класс с псевдоклассом **:not()**. Это псевдокласс исключение – в качестве параметров можно указать другие селекторы, например классы, теги, id или другие псевдоклассы. В данном случае мы указали применять стили ко всем элементам с классом recent-item, кроме последнего в блоке (чтобы отступы и разделитель не отображались у последней записи).



```
/* все кроме последнего с таким классом */
.recent-item:not(:last-child) {
  margin-bottom: 25px;
  border-bottom: 1px solid lightgray;
  padding-bottom: 15px;
}
.recent-item p {
  color: #454546;
  font-weight: 600;
  margin-bottom: 15px;
  cursor: pointer;
  transition: color 250ms ease;
}
.recent-item p:hover {
  color: #ce416e;
}
.recent-item span {
  color: #454546;
  font-size: 14px;
}
```

Рис. 133 – Стили для блога «популярных статей»

В результате наша страница должна быть похожей на рисунок 134. Что касается мобильной вёрстки, то принято на мобильных диагоналях боковое меню и основной контент размещать друг под другом. Добавим для медиа-запроса 992px классу **blog-container** всего два стиля:

max-width: 600px;

flex-direction: column;

Тогда колонка будет не очень широкой на планшетах или занимать всю ширину страницы на мобильных устройствах. А благодаря тому, что элементы расположились в колонку, они не будут друг другу мешать.

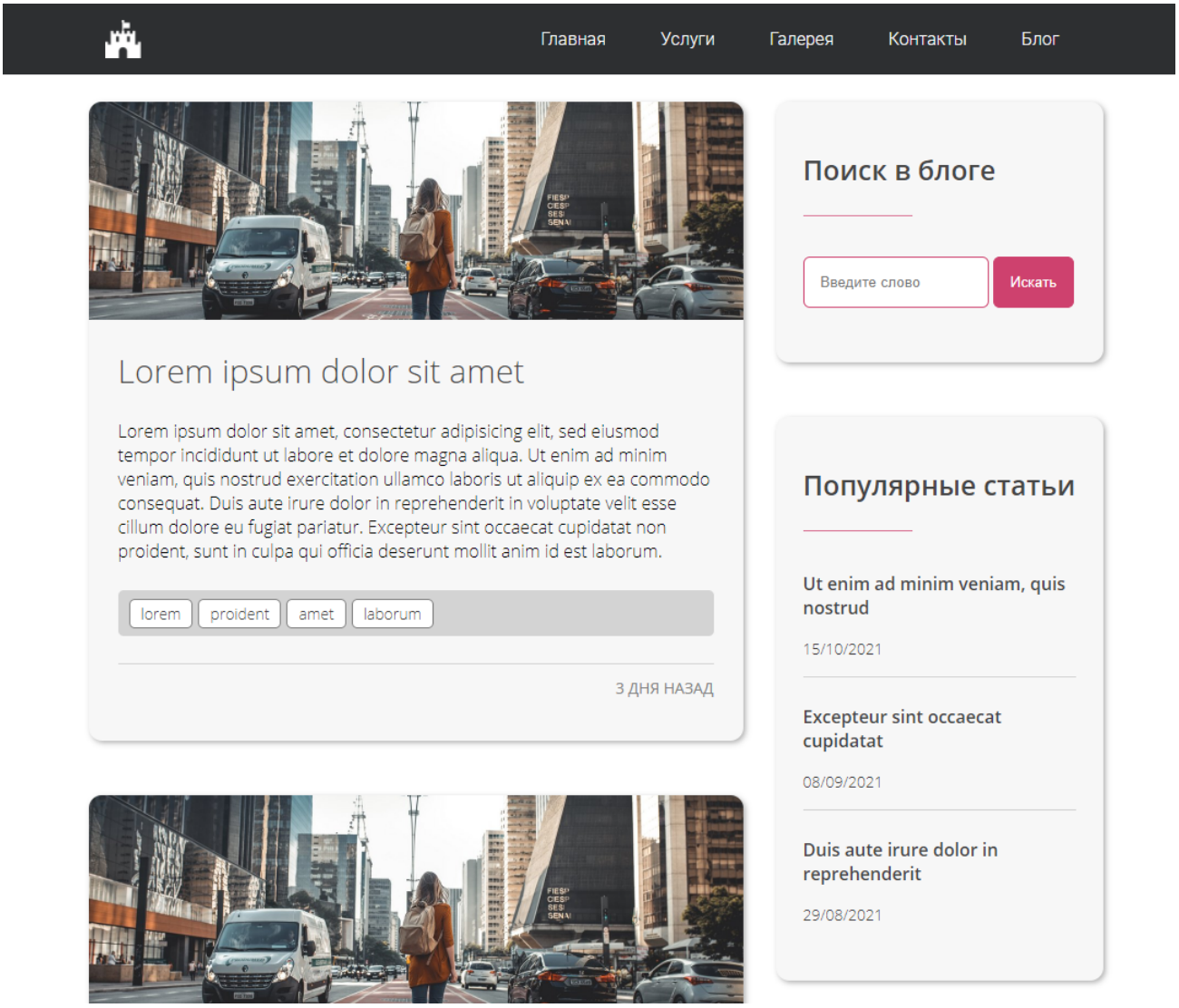


Рис. 134 – Страница «Блог»

Но записи в блог не добавляют вручную через добавление текста прямо в HTML-код. Как правило такие элементы выглядят как шаблон, куда подставляется текст, хранящийся в базе данных, а затем отображается на странице. Сегодня популярны два варианта отображения данных с сервера:

1. Отрисовка данных при формировании страницы на сервере.
2. Отрисовка страницы и заполнение данных после её загрузки.

Представим, что у нас есть сервер, возвращающий данные о записях блога в формате JSON. Тогда мы можем передать эти данные нашему JavaScript, а тот в свою очередь подставит данные на страницу.

Создадим новый файл blog.js и подключим к нашей странице. В нём создадим переменную, в которую запишем данные в формате JSON. Такие данные мы можем получить от сервера через API или при AJAX-запросе.

Сама JSON строка расположена в текстовом файле с дополнительными материалами к работе. Там же лежит строка с HTML разметкой карточки записи блога. Для неё мы также создадим переменную в файле blog.js.

Рис. 135 – Переменные с данными из файла

Запишем непосредственно код отображения записей. Создадим функцию jQuery, ожидающую загрузку страницы, аналогично предыдущей лабораторной работы для файла popup.js.

В ней создадим переменную data в которую запишем объект, полученный путём разбора строки JSON. Разбор строки осуществляется встроенным в JavaScript методом **JSON.parse([строка])**. Полученный объект передадим в качестве параметра функции **drawCards**(). Ниже, после инициализирующей функции создадим функцию **drawCards**() ожидающую параметр с объектом данных для блога.

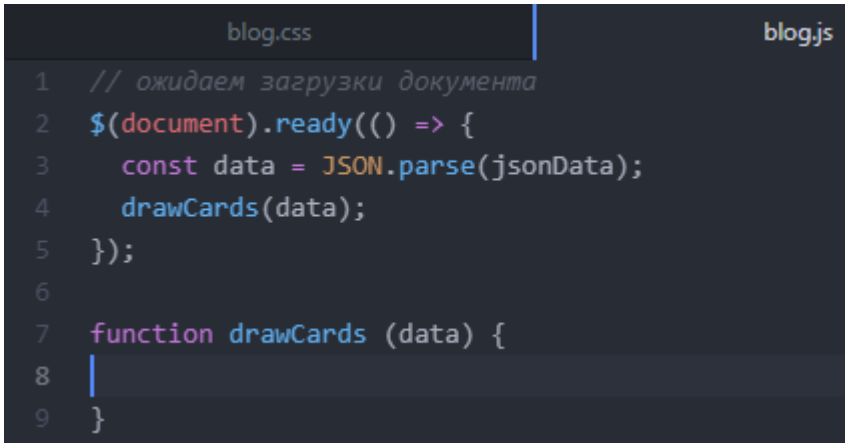


Рис. 136 – Объявление инициализирующей функции и функции «рисования»

Внутри функции **drawCards**() вызовем метод **html**() функции jQuery с пустой строкой для контейнера с записями блога. Таким образом, очистим всё то, что хранится в этом контейнере. Затем обойдём циклом **forEach**() все элементы объекта data.

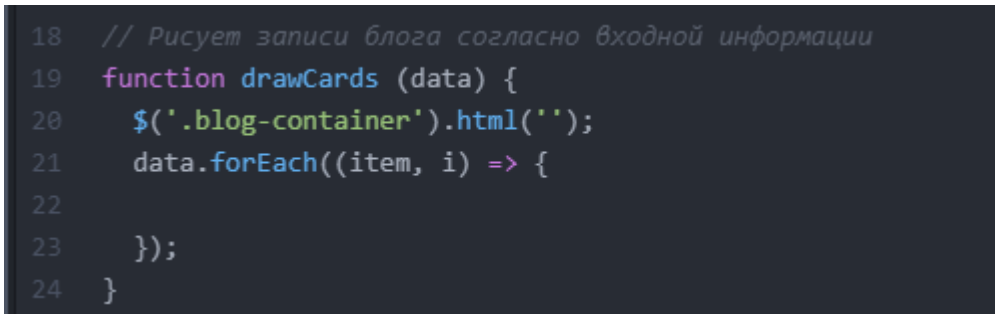


Рис. 137 – Очистка родительского контейнера и цикл по элементам объекта

Здесь, внутри цикла для каждого элемента массива данных создаём копию карточки блога. Для этого создаём переменную, в которую передаём объект jQuery со строкой HTML разметки записи блога. Откуда взять разметку? Мы её уже передали в переменную **cardHtml**, поэтому воспользуемся ей.

Мы могли взять эту разметку и с самой HTML страницы. Но чуть ниже вы поймёте, почему мы не пошли этим путём.

Рис. 138 – передача в переменную данных JSON

Пока на странице ничего не добавилось, ведь мы только создали переменную, но не нарисовали её содержимое в HTML. Для этого мы можем воспользоваться методом **html**(), но он будет перезаписывать одну карточку в контейнер. А нам нужно выводить все, что есть. Для этого удобнее будет метод **append**(). Он схожим образом добавляет **html**() не перезаписывает содержимое, а добавляет его в конец искомого элемента.

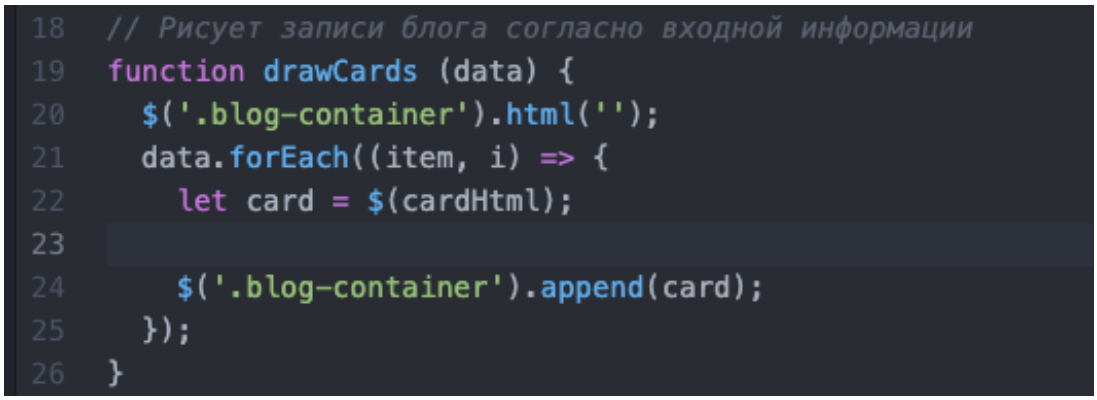


Рис. 139 – метод «дорисовывания» элементов в HTML-тег

Теперь у вас должны отобразиться 4 одинаковые записи блога. Но идея была в том, чтобы применять полученные данные и отображать на странице. Для этого к каждой записи-карточке подставим нужные значения прямо внутри цикла перед тем, как их нарисовать на странице. Тут нам поможет jQuery предоставив удобный инструмент поиска тегов и классов, а также методов вставки данных в них.

```
// Рисует записи блога согласно входной информации
function drawCards (data) {
  $('.blog-container').html('');
  data.forEach((item, i) => {
    let card = $(cardHtml);
    card.find('.blog-cover').css('background-image', 'url("img/blog/' + item.image + '")')
    card.find('.blog-title h2').text(item.title);
    card.find('.blog-text p').text(item.text);
    card.find('.blog-published-date').text(item.date);

    $('.blog-container').append(card);
  });
}
```

Рис. 140 – передача данных из объекта в запись блога

Теперь на странице рисуются записи с разными картинками, заголовками и датами. Для тегов внутри нашего цикла нужно создать ещё один forEach(), чтобы для каждого тега «нарисовать» элемент списка с ссылкой:

```
// Рисует записи блога согласно входной информации
function drawCards (data) {
  $('.blog-container').html('');
  data.forEach((item, i) => {
    let card = $(cardHtml);
    card.find('.blog-cover').css('background-image', 'url("img/blog/' + item.image
    card.find('.blog-title h2').text(item.title);
    card.find('.blog-text p').text(item.text);
    card.find('.blog-published-date').text(item.date);
    let tags = '';
    // обходим все теги объекта
    item.tags.forEach((tag, i) => {
      // создаём элемент списка
      tags += '<li><a href="' + tag + '">' + tag + '</a></li>';
    });
    //добавляем все элементы в родительский <ul>
    card.find('.blog-tags ul').html(tags);

    $('.blog-container').append(card);
  });
}
```

Рис. 141 – обход циклом всех тегов одной записи

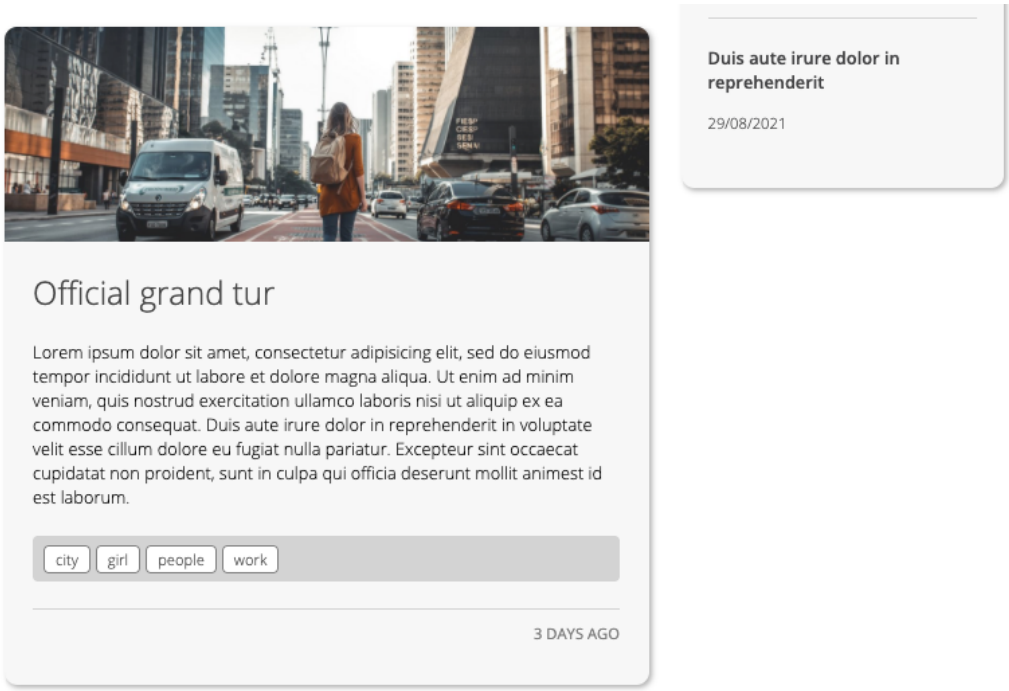


Рис. 142 – запись блога, полученная из данных JSON

Теперь, когда мы можем динамически формировать список отображаемых статей, мы можем пойти дальше и манипулировать этим списком. Например, осуществлять поиск или фильтровать по тегам.

На рисунке 137 мы условились передавать в функцию рисовки данных для отображения. Следовательно, если перед этим данные отфильтровать, то скрипт нарисует только те карточки, что нужно. Напишем функцию поиска.

Для начала создадим обработчик клика по кнопке поиск:

blog.html | blog.js

```
// ожидаем загрузки документа
$(document).ready(() => {
  const data = JSON.parse(jsonData);
  drawCards(data);

  $('.search-do').on('click', () => {
    const search = $('.search-text').val().toLowerCase();
    filter(search, data);
  });
});
```

Рис. 143 – инициализация щелчка мыши по кнопке поиска

Здесь по клику на кнопку «Искать» мы кладём в переменную **search** значение из поля с классом **search-text**, и приводим это значение к нижнему регистру букв для исключения зависимости поиска от входного регистра.

А в восьмой строке вызываем функцию **filter**(), которой передаём поисковый запрос и исходный объект с данными.

```
41  // фильтруем исходный объект по условию
42  function filter(value, data) {
43      const newData = data.filter((item) => {
44
45      })
46      drawCards(newData);
47  }
```

Рис. 144 – передача в переменную данных JSON

Напишем и саму функцию фильтрации. Здесь создаём переменную, в которую будут попадать те элементы, которые подошли условию встроенного метода **filter**. Условием будет поиск совпадений текста в полях объекта.

Метод filter проходит по всем полям объекта как по массиву, таким образом мы можем получить значения каждого его поля и сравнить с искомым запросом. А затем вызовем нашу функцию **drawCards** которой передадим оставшиеся данные.

```
40  // фильтруем исходный объект по условию
41  function filter(value, data) {
42      const newData = data.filter((item) => {
43        let result = 0;
44        result += item.image.toLowerCase().indexOf(value) > -1;
45        result += item.title.toLowerCase().indexOf(value) > -1;
46        result += item.text.toLowerCase().indexOf(value) > -1;
47        result += item.date.toLowerCase().indexOf(value) > -1;
48        result += item.tags.filter((tag) => {
49          return tag.toLowerCase().indexOf(value) > -1;
50        }).length;
51        return result > 0;
52      })
53      drawCards(newData);
```

Рис. 145 – фильтрация данных в исходном объекте по полученному запросу

На рисунке 145 внутри функции была создана проверка. Для поиска текста используется метод **indexOf**(), который проверяет наличие в строке другой подстроки. Если совпадений нет – он возвращает -1, в противном случае индекс символа, с которого начинается совпадение. Мы наращиваем счётчик каждый раз, когда совпадение найдено, таким образом, если хотябы раз было найдено совпадение, то счётчик больше 0 и элемент итерации попадает в переменную newData.

Обратите внимание, что здесь мы также при проверке вызываем метод **toLowerCase**(), чтобы сравнивать входные данные в одном регистре.

Теперь вводя в поле поиска уникальный запрос будет рисоваться только те записи, которые подходят.

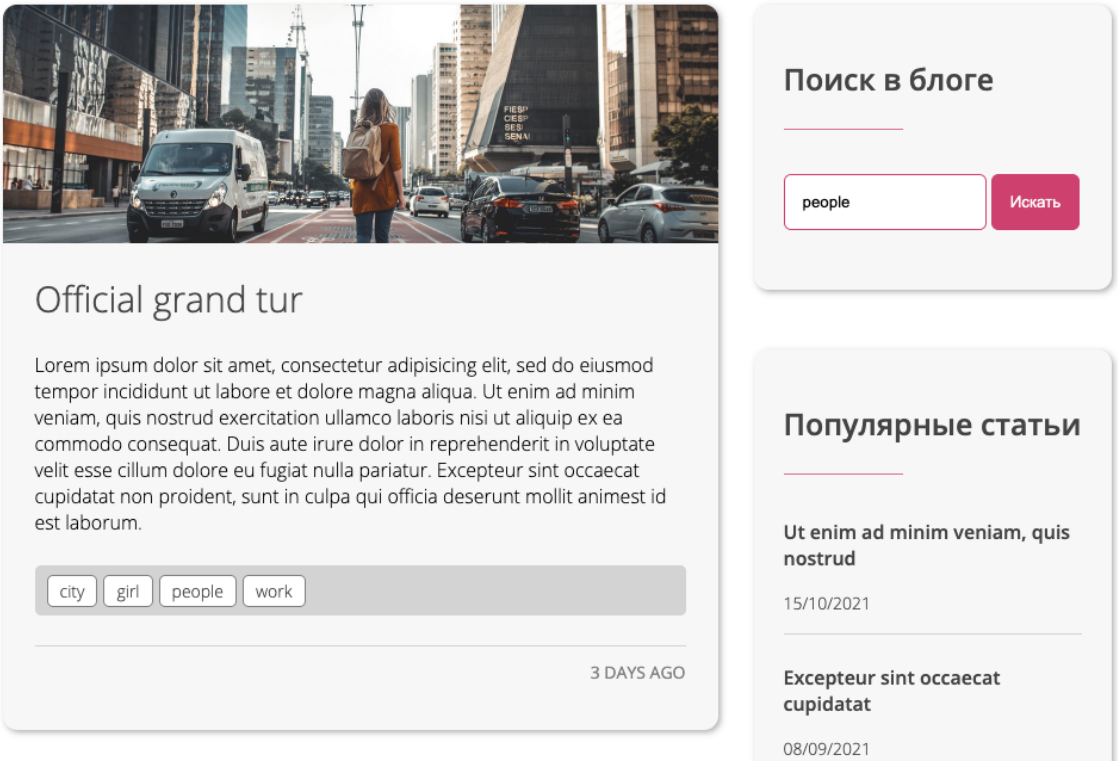


Рис. 146 – отображение результата поиска

Теперь, когда есть функция фильтрации и рисования отфильтрованных данных, создать поиск по тегам становится очень просто. Необходимо только обработать клик по тегу и передать функции поиска запрашиваемое значение.

```
    $('.search-do').on('click', () => {
      const search = $('.search-text').val().toLowerCase();
      filter(search, data);
    });

    $('.blog-tags a').on('click', (e) => {
      e.preventDefault();
      const search = $(e.currentTarget).text().toLowerCase();
      filter(search, data);
    });
  });
```

Рис. 147 – обработка щелчка мыши по тегу

Всё остальное наш код уже умеет делать. Один важный момент. После поиска или фильтра по тегам, клик по тегам больше не работает. Это происходит по тому, что мы удалили карточки, к которым было привязано событие клика. После того, как были нарисованы карточки необходимо снова назначить событие клика на теги, которые только что были нарисованы с новым HTML кодом.

В таком случае мы можем вынести функцию инициализации клика по тегам отдельно. И вызывать её каждый раз ПОСЛЕ того, как были нарисованы записи.

```
$(document).ready(() => {
  const data = JSON.parse(jsonData);
  drawCards(data);

  $('.search-do').on('click', () => {
    const search = $('.search-text').val().toLowerCase();
    filter(search, data);
  });

  initCardsHandler();
});

// инициатор кликов по тегам
initCardsHandler () {
  $('.blog-tags a').off().on('click', (e) => {
    e.preventDefault();
    const search = $(e.currentTarget).text().toLowerCase();
    filter(search, data);
  });
}
```

Рис. 148 – передача в переменную данных JSON

На рисунке 148 показан пример, как должен работать код при начальной отрисовке страницы. Подумайте, где нужно создать вызов этой функции, чтобы клик по тегам работал после поиска.

Задание на контроль

1. Повторить пример из практической части.

2. Создать страницу «Блог», добавить на неё ссылку в меню других страниц и разметку из примера.

3. Создать новый файл стилей для страницы.

4. Создать скрипт динамического отображения записей блога.

5. Создать логику поиска записей.

6. Добавить фильтрацию по тегам.

7. Лабораторная работа считается защищенной, если студент показал в окне браузера страницу из практической части со стилизованным меню и ответил правильно на все заданные контрольные вопросы (следующая лабораторная работа не подлежит защите до тех пор, пока не будет защищена предыдущая).