Лабораторная работа № 1. HTML

Подготовка к работе

Для работ требуется среда разработки, рекомендуется VS Code, далее все примеры будут рассмотрены для данной среды. VS Code из коробки имеет базовую поддержку HTML – подсветка кода, автоматическое закрытие тегов контейнеров, предложения завершения кода IntelliSense.

Добавьте в рабочую среду папку, в которой желаете хранить файлы работ (File – Add folder to workspace), создайте в ней файл (для этого нажмите правой кнопкой мыши, далее ПКМ на имя папки в боковом меню explorer, выберите New File и введите желаемое имя файла с расширением .html на конце).

Задания для самостоятельного выполнения

Общая информация по проекту: вы создаете страницу для компании, продающей компьютерные комплектующие (процессоры, материнские платы, видеокарты). Всего проект должен содержать 5 страниц. В title каждой из страниц после названия компании добавляется «–» и название самой страницы (кроме Главная). Сайт должен содержать главную страницу с общей информацией о предприятии, страницу с ценами на продаваемые комплектующие, страницу новостей техники и технологий (или самого предприятия, на ваше усмотрение), страницу контактов с почтой, телефоном и адресом и картой, указывающей местоположение предприятия.

1. Создайте страницу html по макету в табл. 1.1:

1.1) верхняя часть (выделена серым) – header, средняя (контент) – div, нижняя – footer;

1.2) в header логотип – div, нижняя часть (меню – главная, прайс-лист, …) должен быть таблицей из 1 строки на 4 ячейки;

1.3) таблица в header должна быть она была растянута на весь экран по ширине;

1.3) в качестве логотипа подберите из интернета подходящий, по вашему мнению, файл png, сохраните его под именем logo.png в папке с работой, добавьте его на страницу в то место, где в табл. 1.1 написано Логотип;

1.4) title страницы – имя компании, придумайте свое;

1.5) в качестве разработчика (footer) укажите себя, в начале добавьте знак копирайта;

1.6) сохраните файл под именем index.html.

Таблица 1.1 – Макет страницы

|  |  |  |  |
| --- | --- | --- | --- |
| Логотип | | | |
| Главная | Прайс-лист | Новости | Контакты |
| Контент | | | |
| Информация о разработчике, год создания | | | |

2. Полученная страница является прототипом для всех остальных страниц проекта. Сделайте элементы меню ссылками на страницы со следующими именами файлов:

Главная – index.html;

Прайс-лист – price.html;

Новости – news.html;

Контакты – contacts.html.

3. Продублируйте полученный файл index.html 3 раза. Присвойте копиям имена, указанные в прошлом пункте.

4. Модифицируйте полученные файлы:

4.1) в полученных страницах, кроме index.html, добавьте в title в конце «–» и название самой страницы, аналогичное тому, что указано в панели меню header (Название компании – Прайс-лист и т.п.);

4.2) добавьте в каждой странице метатег с ключевыми словами (по 2 ключевых слова для каждой страницы).

5. Наполните содержимым главную страницу:

5.1) контент главной должен содержать 2 параграфа, изображение (png/jpg), видео (mp4/webm), подходящих к деятельности компании, список видов продукции;

5.2) в div сперва должен идти параграф текста на 50-100 слов;

5.3) затем изображение;

5.4) затем второй параграф текста на 50-100 слов;

5.5) затем видео;

5.6) затем параграф с текстом «Перечень реализуемой продукции:» и следующий за ним ненумерованный список из 3 текстовых элементов: процессоры, материнские платы, видеокарты.

6. Наполните содержимым страницу Прайс-лист:

6.1) на странице в div должны располагаться 3 таблицы, каждая таблица должна иметь 2 столбца, 6 строк, границы, заголовок над таблицей;

6.2) заголовок (шапка) таблиц должен быть выполнен тегом thead;

6.3) каждая таблица является прайс-листом на 6 популярных сейчас товаров одной из категорий: процессоры, материнские платы, видеокарты;

6.4) между таблицами должен присутствовать отступ в 2 переноса строки;

6.5) в конце страницы, после таблиц, в отдельном div должен присутствовать параграф текста: «Цены указаны в рублях для физических лиц на момент текущая\_дата». Вместо текущая\_дата должна стоять дата выполнения работы.

7. Наполните содержимым страницу Новости:

7.1) страница должна содержать 5 новостей либо о вашей компании, либо новостей о технике и технологиях, либо их комбинацию;

7.2) каждая новость выполняется по следующей структуре (в скобках указаны правила форматирования):

Название новости *(заголовок h2)*

Дата новости (параграф, курсив)

(изображение высотой 200, шириной 300)

Изображение

Текст новости (параграф текста 50-100 слов)

7.3) каждая новость включена в контейнер article;

7.4) каждая новость – таблица из 4 строк (1 для названия, 2 для даты, 3 для изображения, 4 для текста) с шириной 400;

7.5) между новостями должен присутствовать отступ в 2 переноса строки.

8. Наполните содержимым страницу Контакты:

8.1) выберите реальный адрес в городе России;

8.2) перейдите на сайт <https://yandex.ru/map-constructor/> для создания карты;

8.3) найдите на сайте выбранный адрес, нажмите кнопку ;

8.4) нажмите левой кнопкой мыши (далее ЛКМ) на место расположения на карте выбранного адреса;

8.5) заполните поле Подпись метки и нажмите Продолжить;

8.6) в левом меню нажмите ;

8.7) разместите появившуюся на карте рамку так, чтобы ваша метка располагалась по центру, сожмите, при желании, рамку карты;

8.8) нажмите , затем  и скопируйте себе появившийся код вставки карты;

8.9) полученный код разместите в div страницы Контакты;

8.10) под картой расположите параграф с полным адресом выбранного места (включая индекс).

Пояснения к выполнению

Для выполнения работы вам понадобится знание тегов из лекционного материала. Кратко сведения о тегах представлены ниже.

Теги могут содержать атрибуты, меняющие параметры форматирования содержимого тега:

<тег атрибут1="значение" атрибут2="значение">

<тег атрибут1="значение" атрибут2="значение">...</тег>

Парные теги также называют контейнерами. Порядок атрибутов не важен.

Базовая структура HTML приведена в табл. 1.2.

Таблица 1.2 – Базовый код HTML

|  |
| --- |
| Исходный код HTML |
| <!doctype html>  <html>  <head>  <meta charset="utf-8" />  <title>Моя тестовая страница</title>  </head>  <body>  <img src="img/img.png" alt="Изображение" />  </body>  </html> |

<!DOCTYPE html> - указание типа документа, требуется для корректной проверки ошибок валидатором или браузером. Если не указать, могут быть использованы устаревшие стандарты при проверке ошибок, для html5 следует указывать этот тег.

Документ отражается последовательной обработкой всех тегов по порядку.

<html></html> – корневой элемент страницы html. Все прочие теги и весь контент должен располагаться в нем.

<head></head> – контейнер для содержимого веб-документа, не являющегося контентом – скрипты, заголовок страницы на вкладке в браузере, ключевые слова и описание страницы, стили CSS и др.

<meta charset="utf-8"> - метатег, используется для передачи сведений браузеру информации обо всем документе (тип документа, ключевые слова, заголовки, …), в табл. 1 – кодировку текста. Например, метатег, содержащий ключевые слова для поисковой системы duckduckgo, мог бы выглядеть следующим образом: <meta name="keywords" content=“duckduckgo, поисковая система" />.

<title></title> - заголовок страницы, отображается как загловок вкладки в браузере, при сохранении страницы в закладки и т.п.

<body></body> - основное содержимое страницы – текст, медиафайлы, таблицы, скрипты и т.д.

<img src="img/img.png" alt="Изображение" /> - добавляет на страницу изображение. Атрибуты: src – путь к файлу, alt – текст, отображаемый вместо изображения, если файл недоступен или включен режим для слабовидящих и т.п., title – текст, отображаемый при наведении мышью на изображение.

height="10" – высота изображения в пикселах (здесь 10), width="10" – ширина изображения в пикселах (здесь 10). Если указана только height/width – второй из параметров берется таким, чтобы сохранить исходные пропорции изображения. decoding="тип" – декодирование разных типов: sync – синхронно с загрузкой остального контента, async – асинхронно (в фоне параллельно другим элементам), auto – решает браузер. loading="тип" – загрузка, если тип eager – сразу же, независимо от того, в области ли оно просмотра, lazy – при приближении области просмотра к изображению.

Теги заголовков:

<h1>Главный заголовок</h1>

<h2>Заголовок верхнего уровня</h2>

<h3>Подзаголовок</h3>

<h4>Под-подзаголовок</h4>

Доступны 6 уровней заголовков, рекомендуется использовать не более 4.

Браузер способен отобразить страницу с отдельными тегами, даже без html, body (как 4 строки выше), в этом случае браузер сам исправляет ошибку, подставляя эти теги при рендере.

Основные текстовые теги:

<p>текст</p> - одиночный абзац текста с переносом строки в конце;

<ul></ul> - ненумерованный список;

<ol></ol> - нумерованный список;

<li></li> - элементы списка, вложены в <ul> или <ol>.

Теги для форматирования текста:

<i></i> - курсив;

<b></b> - полужирный.

Теги нижнего подчеркивания u считаются устаревшими с выходом html5, рекомендуется оформлять текст стилями CSS.

<a href="ссылка">текст ссылки</a> - кликабельная гиперссылка по заключенному в контейнер тексту, аргумент href – ссылка*.*

В тег ссылки можно заключить не только текст, например, изображение: <a href="ссылка"><img src="путь"></a>.

<br> - одиночный тег, добавляющий перенос строки между элементами. *К примеру, если вы вставите два изображения подряд, они будут отображаться в ряд. Чтобы перенести второе изображение на новую строку может пригодиться данный тег. Также работает с текстовыми и другими элементами*.

Таблица строится из контейнеров:

<table> - таблица;

<caption> - заголовок над таблицей;

<thead> - заголовок (шапка) таблицы;

<tbody> - основная часть таблицы;

<tr> - строка таблицы (в т.ч. заголовка);

<th> - заголовок столбца (в thead);

<td> - ячейка таблицы.

Пример составления таблицы 4х4 с двумя заголовками приведен на рис. 1.1. Конечный элемент заголовка – th, а тела таблицы – td. Оба они являются, по сути, ячейками.

Аргументы width, height применимы и к тегу table. width="100%" растягивает таблицу на всю ширину окна. border= "число" добавляет границу к таблице, устаревший тег, рекомендуется использовать CSS для смены оформления таблиц.

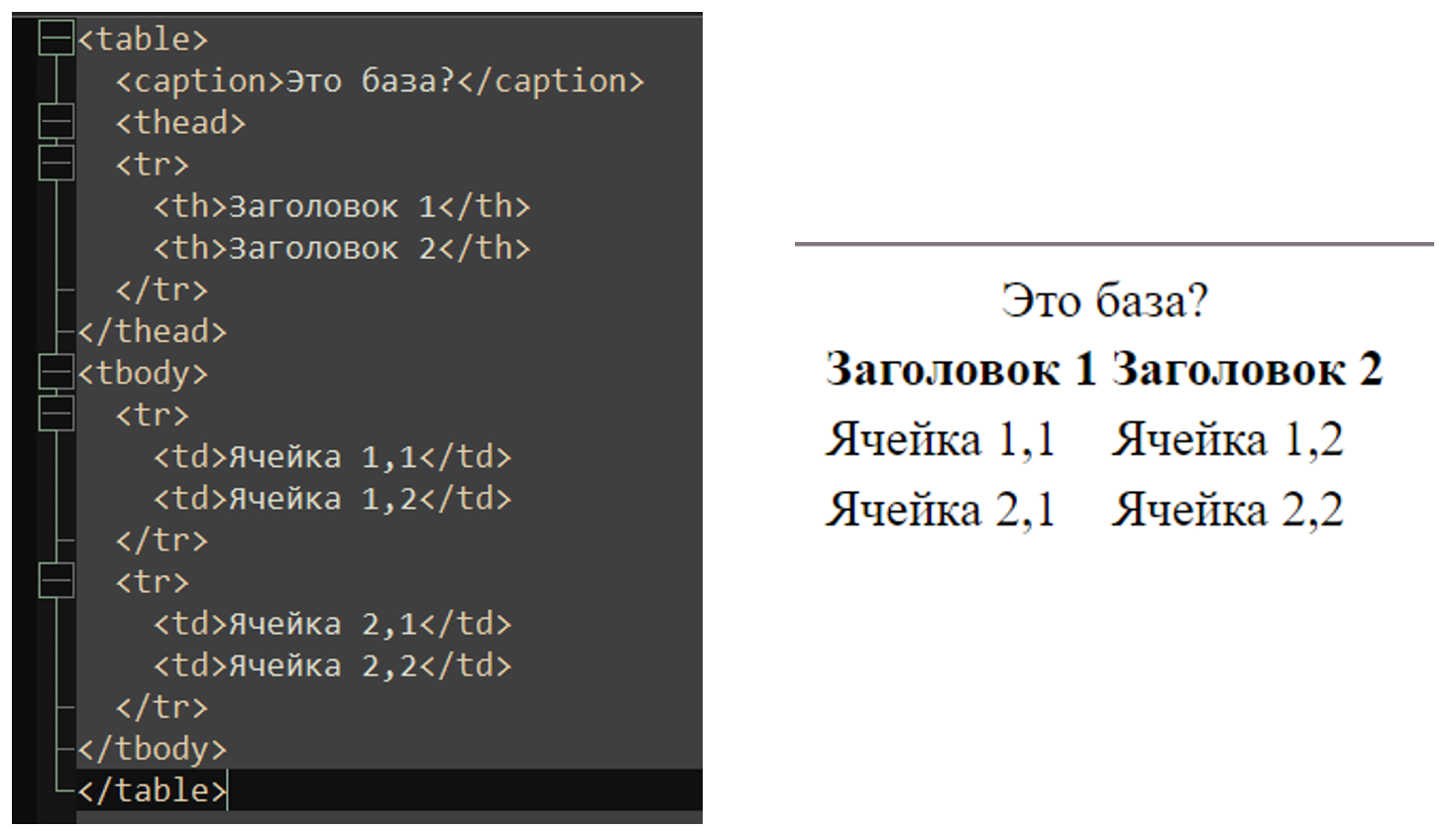


Рис. 1.1 – Пример таблицы

Также в страницы html можно встраивать другие страницы, регулируя размер рамки, по которой они вставляются. <iframe> - встраивание другой страницы.

Аргументы: width, height – размер, src – ссылка на страницу.

Можно использовать для ускорения работы с сайтом, при работе с областью iframe будет обновляться только ее часть (см. рис. 1.2).



Рис. 1.2 – Код iframe

По умолчанию вызываемый iframe ресурс может иметь слишком много прав (выполнять скрипты и т.д.). Это дает простор для возможных проблем безопасности, поэтому следует ограничить возможности iframe. Чтобы ограничить права iframe нужно в его тег добавить аргумент sandbox.

<div> - контейнер для отдельного блока страницы, используется, чтобы применять к разным блокам различное оформление, скрипты и т.д.

<span> - контейнер для присвоения стилей элементом внутри строчных элементов (параграф, ячейка, …).

В HTML5 для удобства организации структурных частей body введены теги контейнеров, разделяющих их по смыслу (сами не меняют форматирование, если использовать без аргументов):

<header> - блок в верхней части страницы (меню, заголовок, …);

<article> - содержимое с самодостаточным контентом, к примеру, статья, отзыв и т.д.;

<aside> - блоки дополнительной информации – реклама, ссылки по теме статьи, …;

<section> - отдельный по теме блок страницы;

<nav> - блок навигации по странице;

<footer> - контактная и пр. информация, авторские права и т.д., как правило внизу сайте.

Преимущество этих контейнеров перед div в лучшей читаемости кода и большим вниманием поисковиков к header, footer, …, чем к div.

Большим преимуществом HTML5 является введение тегов для мультимедиа, раньше для его вставки требовались сторонние плееры на Adobe AIR, Flash или Microsoft Silverlight, а для векторной графики иные костыли. Новые теги для мультимедиа в HTML5:

<svg> - вставка векторных файлов svg;

<audio> - вставка аудио;

<video> - вставка видео.

Аргументы video, audio: src – источник, controls – вывести элементы управления

Пример кода для добавления видео и аудио на страницу приведен на рис. 1.3.

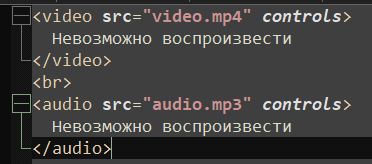


Рис. 1.3 – Пример медиа тегов

Лабораторная работа №2. Оформление web-страниц

Подготовка к работе

Для выполнения работы необходимо загрузить себе некоторые медиа ресурсы из архива по ссылке:

<https://drive.google.com/file/d/1ROGI-Pz1T66E69NjeX6FJ85xNIUg-E9d/view?usp=sharing>.

Далее в тексте пояснений будут указаны имена фалов из данного архива. Изображения расположены в папке img, видео в папке video.

Задания для самостоятельного выполнения

В данной работе освоение навыков декорирования документа CSS планируется на примере создания сайта фотографу. Сайт состоит из одной страницы с несколькими секциями.

1) Создайте страницу index.html по шаблону в таб. 2.1, соблюдая правила для структурных частей, помеченных в правом столбце (данный столбец не относится к макету страницы):

(меню в header)

1.1) часть (1) – блок header с фиксированным меню;

1.2) высота (1) – 120px, padding размером 10px задний фон –цвет по желанию.

Чтобы добавить код цвета в каком-либо месте, нажмите на клавиатуре Ctrl+Shift+P для активации поиска инструментов, введите с клавиатуры Color Picker, подтвердите выбор инструмента по Enter – появится окно выбора цвета, после выбора нужного нажмите Insert. При следующем нажатии Ctrl+Shift+P инструмент выбора цвета останется первым в списке, можно не вписывать его имя в строке, а просто нажать Enter для активации;

1.3) также для (1) нужна тень со сдвигом по горизонтали 0px, по вертикали 10px, размытием 10px, размером тени 10px и черным цветом (#000);

1.4) логотип – изображение в блоке img с классом logo, тип отображения в CSS – block, файл img/logo.png, высота auto, width: 50% (чтобы изображение меняло размер при изменении размера окна) и min-width: 450px, чтобы оно не становилось слишком мелким;

1.5) после изображения в теге menu содержатся 2 ссылки (тег a) с текстом Наверх, Отзывы, Об авторе, Галерея и Заказать, параметры href у ссылок оставьте пока пустыми;

1.6) в CSS у menu должен быть задан внутренний отступ padding со значением 0%, размер шрифта large, толщина букв bold, а также position: fixed и отступ сверху top:80px, чтобы при изменении размера логотипа меню не перемещалось;

1.7) ссылкам в CSS требуется изменить цвет всех трех состояний (link, hover, visited): у link и visited цвет должен совпадать с логотипом (#000), у hover быть светлее *(при вводе свойства color в VS Code вам будет предложен список стандартных цветов – выберите любой, затем нажмите на появившийся квадрат с цветом перед названием выбранного цвета, чтобы активировать окно ручной регулировки цвета)*;

1.8) у состояний ссылок link и visited не должно быть выделения нижним подчеркиванием (text-decoration в CSS), а у hover должно быть underline;

(body)

1.9) часть страницы (2) на данном этапе не настраивается, добавьте в body пустой блок main;

1.10) для body в CSS нужно задать параметры, позволяющие сделать его адаптивным к размеру окна (min-height: 100vh; position: relative; margin: 0 0 0 0);

1.11) также для body задайте background-color произвольным цветом, также выберите семейство шрифтов (любое из предлагаемых VS Code после ввода «font-family:»);

(footer)

1.12) в части страницы (3) разместите footer высотой 50px;

1.13) у footer должна быть задано положение внизу, для чего в CSS нужно задать position: absolute (точное положение по координатам), и отступы края блока footer от конкретных краев окна – left: 0 и bottom: 0, также требуется выставить ширину 100%;

1.14) у footer задайте background-color совпадающий с header;

1.15) в main, после header и перед footer должны быть невидимые блоки с классами invis-header и invis-footer, для этих блоков в CSS нужно применить высоту, равную высоте header (140px – 120+10\*2, где 10 – отступы padding сверху и снизу) и footer соответственно;

1.16) в содержимом footer поставьте знак копирайта (©), свое ФИО, год.

Таблица 2.1 – Макет общего шаблона

|  |  |
| --- | --- |
| Логотип  Наверх Отзывы Об авторе Галерея Заказать | (1) |
| main | (2) |
| ©Ваше ФИО, 2024 | (3) |

2) Добавьте адаптивное изображение:

2.1) в main в область (2) добавьте адаптивное изображение в блоке picture на 2 источника: img/adaptive-full.avif с условием (min-width: 601px), img/adaptive-cropped.avif с условием (max-width: 600px), а также img/adaptive-full.jpg для изображения по умолчанию, в alt опишите парой слов содержимое изображения;

2.2) добавленному изображению (тегу img) задайте класс adaptive, а в CSS задайте изображению по данному тегу параметры для автоматического пропорционального изменения размера по ширине окна браузера.

3) Добавьте ***отзывы:***

3.1) после изображения добавить section с классом reviews, в его начале заголовок h2 с текстом «Отзывы», после которого div с классом reviews-row, что соответствует части (3) в табл. 2.2;

3.2) внутри div разместить 2 блока div с классом review и двумя параграфами – первый с классом name, второй с классом reviews-text;

3.3) внутри каждого p разместить span с открывающей кавычкой и пробелом в качестве содержимого (скопируйте отсюда:« ), присвоить span класс quote;

3.4) в CSS классу reviews-row задайте выравнивание текста (text-align) по центру, что позволит выровнять и inline-block;

3.5) классу name задайте размер шрифта medium, и толщину букв bold и стиль шрифта italic;

3.6) классу review-text задайте стиль шрифта italic, а классу quote – размер шрифта xx-large;

3.7) классу review задайте отображение inline-block, ширину 80%, отступы внутренние, наружные по 5 px, размер шрифта medium;

3.8) добавьте в CSS условие @media на минимальную ширину экрана 600px, в нем для класса reviews-row задайте отображение flex, justify-content по центру и ширину 100%, а классу review – ширину 40% и высоту auto, все это позволит при большой ширине экрана выводить отзывы в 2 столбца, а по умолчанию они выводятся в 1;

3.9) продублируйте готовый div с классом reviews-row вместе с содержимым (вставьте код сразу после конца этого div), наполните отзывы содержимым (имена и текст отзывов) по своему усмотрению или возьмите готовые из файла reviews.txt.

4) Добавьте раздел ***об авторе:***

4.1) внутри main после section с классом reviews добавьте section с классом author, который будет соответствовать части страницы (4) из макета в таблице 2.2;

4.2) в начале блока добавьте заголовок h2 с текстом «Об авторе», после которого фотографию автора img/author.png;

4.3) после фотографии добавьте параграф, в начало которого скопируйте span с открывающей кавычкой из review;

4.4) элементам section, img и p из данного раздела присвойте класс author;

4.5) добавьте в параграф после span текст описания себя фотографом – произвольный или взятый из файла bio.txt;

4.6) блоку section с классом after в CSS задайте ширину 100% И margin-bottom в 40px;

4.7) для всех наследников блока section класса author (все наследники выбираются по \*) задайте свойства: margin-top и margin-bottom по 10px, а margin-left и margin-right со значением auto, последнее аналогично margin: auto выровняет элементы section по середине, но позволит задать им фиксированную ширину и/или высоту;

4.8) блоку img класса author задайте отображение типа block и высоту 200px;

4.9) параграфу класса author задайте ширину 70% и минимальную ширину 600px, внутренний отступ 5px, границу и фоновый цвет аналогично классу review.

5) Добавьте ***галерею:***

5.1) внутри main после section с классом author добавьте section с классом gallery, который будет соответствовать части страницы (5) из макета в таблице 2.2;

5.2) скопируйте после заголовка оба div класса review-row из section класса reviews;

5.3) удалите параграфы с классом review-text в копии, замените имена классов: reviews-row на media-row, review на media, а name на title;

5.4) в CSS добавьте (через запятую) к .reviews-row класс .media-row, к .review - .media, к .name - .title;

5.5) после заголовков добавьте фото (верхний media-row) и видео (нижний media-row), используя avif и jpg форматы для изображений (файлы img/gallery1.avif,\*.jpg и img/gallery2.avif,\*.jpg) и webm и mp4 форматы для видео (файлы video/video1.mp4,\*.webm и video/video2.mp4,\*.webm), для видео также включите controls;

5.6) этим фото и видео задайте класс media-content, для которого в CSS задайте правила – ширина 100% и высота auto, чтобы объекты адаптировались к ширине блоков.

6) Добавьте раздел ***заказать:***

6.1) внутри main после section с классом gallery добавьте section с классом purchase, который будет соответствовать части страницы (6) из макета в таблице 2.2;

6.2) в начале блока добавьте заголовок h2 с текстом «Заказать»;

6.3) далее потребуется привести список возможных услуг, выровняв его по центру, чтобы сделать это, необходимо разместить сам список в блок, поэтому после заголовка создайте div с классом list-container;

6.4) внутри div добавьте ненумерованный список <ul> с несколькими элементами <li> внутри (не менее 4) и опишите их или возьмите готовый текст из файла purchases.txt (без последней строки);

6.5) после списка добавьте параграф текста (в том же div) с контактным номером или добавьте последнюю строку из файла purchases.txt;

6.6) чтобы список выравнивался по центру задайте в CSS блоку div класса list-container параметр text-align по центру, а также добавьте отступ от footer через параметр margin-bottom: 60px;

6.7) в CSS задайте ul с классом purchase отображение в виде inline-block, чтобы подействовало выравнивание по центру, а также text-align: left, чтобы отдельные элементы списка выравнивались по левому краю;

6.8) поскольку маркеры списка по умолчанию крайне невзрачны, замените их, для чего в параметрах CSS для ul задайте также list-style: none;

6.9) для замены убранных маркеров на собственные создайте в CSS правило li::before{}, которое добавит указанный в нем элемент сразу перед (слева) элементом li, в этом правиле укажите параметр content со значением в виде символа маркера, например   
content: "💸 "; (можно использовать эмодзи).

7) Разместите ссылки на части страницы:

7.1) в начале каждой section (сразу после открывающего тега) добавьте строку <a id="имя" class="inside-link"></a>, где вместо «имя» соответственно укажите имя секции (reviews, author, gallery, purchase);

7.2) еще одну такую ссылку с именем top разместите в начале main, сразу после блока invis-menu;

7.3) в CSS классу inside-link задайте тип отображения block, положение relative, отступ сверху -140px и visibility: hidden, все это нужно для компенсации смещения объектов страницы из-за фиксированного меню высотой 140px;

7.4) в меню, в тегах ссылок a расставьте ссылки на добавленные объекты, для чего перед их именами (top, reviews, author, gallery, purchase) поставьте символ # - это позволит ссылкам работать внутри страницы;

7.5) чтобы по нажатии на ссылку пролистывание было плавным и не появлялось каретки выделения места перехода, в CSS для тега html задайте scroll-behavior: smooth и caret-color: transparent.

Таблица 2.2 – Макет полного контента страницы

|  |  |
| --- | --- |
| Логотип  Наверх Отзывы Об авторе Галерея Заказать | (1) |
| Адаптивная фотография, на всю ширину окна | (2) |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | ФИО1  Текст отзыва1 |  | ФИО2  Текст отзыва2 |  | |  |  |  |  |  | |  | ФИО3  Текст отзыва3 |  | ФИО4  Текст отзыва4 |  | | (3) |
| |  | | --- | | Текст описания страницы автора | | (4) |
| Галерея   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | photo1 |  | photo2 |  | |  |  |  |  |  | |  | video1 |  | video2 |  | |  | | | | | | (5) |
| Заказать   |  | | --- | | Информация по стоимости и контакты | |  | | (6) |
| ©Ваше ФИО, 2024 | (7) |

Пояснения к выполнению

Для выполнения работы вам понадобится знание лекционного материала. Кратко данные сведения приведены ниже.

Для ручного выбора изображений в зависимости от медиа-условий можно использовать тег picture (табл. 2.3).

Таблица 2.3 – Изображения picture

|  |
| --- |
| Исходный код HTML |
| <picture>  <source media="(max-width: 800px)" srcset="файл1.jpg"/>  <source media="(max-width: 1200px)" srcset="файл2.jpg"/>  <source media="(min-width: 1201px)" srcset="файл3.jpg"/>  <img src="файл\_по\_умолчанию.jpg" alt="Описание"/>  </picture> |

Внутри контейнера picture указываются несколько источников изображений source с соответствующими медиа-правилами. Каждый source может содержать несколько изображений, так как аргумент источника – srcset. Перед закрывающим </picture> должен стоять тег img с изображением по умолчанию, оно используется в устаревших браузерах, не поддерживающих медиа-правила, а также оформления всего блока изображения. min-width – медиа-правило, выполняемое, если размер экрана не более указанного значения.

При размещении изображений через picture можно указывать несколько источников разных типов через source, браузер пользователя выберет наиболее подходящий или покажет содержимое <img>, если ни один из форматов не поддерживается (см. табл. 2.4).

Таблица 2.4 – Изображение с несколькими источниками

|  |
| --- |
| Исходный код HTML |
| <picture>  <source srcset="image.avif" type="image/avif"/>  <source srcset="image.webp" type="image/webp"/>  <img src="image.jpg"/>  </picture> |

Аргумент type должен иметь значение формата выбранного медиафайла, чтобы браузер понимал, поддерживается ли он. Примеры форматов: video/mp4, video/webm, video/ogg, audio/mp3, audio/ogg, audio/aac, image/avif, image/webp, image/gif, image/png, image/svg+xml, image/jpeg.

По аналогии с picture можно указывать несколько источников разных типов, браузер пользователя выберет наиболее подходящий или покажет содержимое <p>, если ни один из форматов не поддерживается (см. табл. 2.5).

Таблица 2.5 – Видео с несколькими источниками

|  |
| --- |
| Исходный код HTML |
| <video controls>  <source src="video.webm" type="video/webm"/>  <source src="video.mp4" type="video.mp4"/>  <p>Браузер не поддерживает webm или mp4</p>  </video> |

Аналогично video можно указывать несколько источников audio (см. табл. 2.6).

Таблица 2.6 – Аудио с несколькими источниками

|  |
| --- |
| Исходный код HTML |
| <audio controls>  <source src="audio.ogg" type="audio/ogg"/>  <source src="audio.mp3" type="audio/mp3"/>  <p>Браузер не поддерживает ogg или mp3</p>  </audio> |

Формат aac, имеет проблемы/особенности с поддержкой, за исключением Safari и современных версий Chrome.

CSS (Cascading Style Sheets) – каскадные таблицы стилей, формальный язык декорирования внешнего вида веб-документов.

Стилизация элемента производится добавлением файла стилей: <link rel="stylesheet" href="styles.css" />. Код добавляется в head. Первый аргумент указывает, что используется CSS, второй – путь к файлу CSS. *Структура файла стилей*:

элемент {

параметр1 : значение;

параметр2 : значение;

}

Вместо «элемент» подставляется имя изменяемого html тега, параметры и значения выбираются из допустимых.

Тегам html можно присваивать классы <тег class="имя"/> чтобы применять стили к элементам с этим классом:

.класс {

параметр : значение;

}

Класс или имя объекта в CSS называют селекторами. Универсальный селектор \* выбирает все объекты документа, объект.\* - все классы в объекте.

Использование классов иллюстрируется на рис. 2.1. Поскольку h1 и p имеют один класс, масштабирование размера шрифта применяется к обоим элементам.

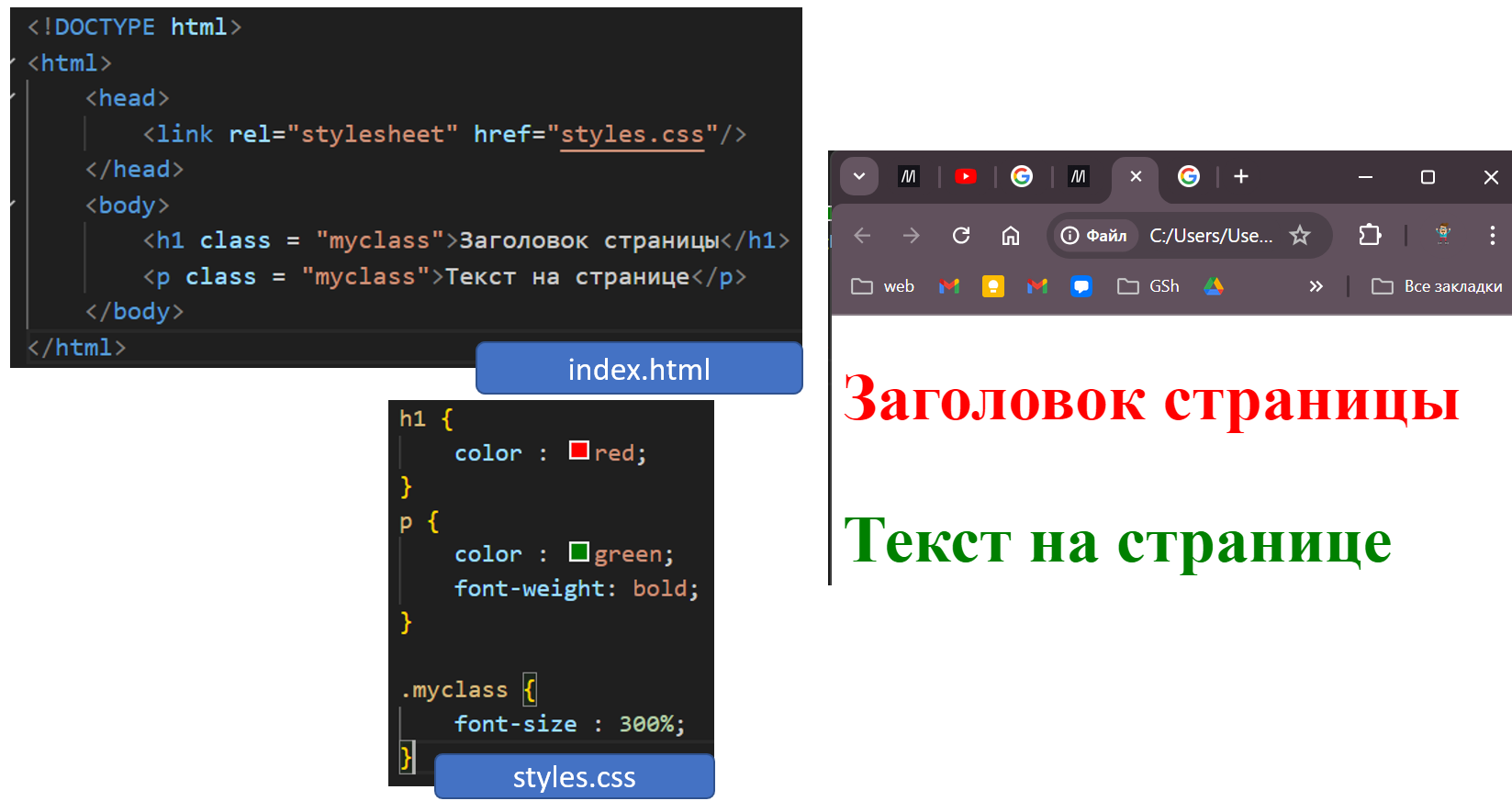


Рис. 2.1 – Пример использования CSS с классом

Для применения свойств только к конкретным объектам (тегам) класса перед «.» указывают имя объекта:

тег.класс {

параметр : значение;

}

Для применения правил к нескольким объектам, их имена следует указывать через «,»:

объект1,

объект2 {

параметр: значение;

}

Для применения правил к объекту2, наследнику объекта1, нужно указать его имя через пробел:

объект1 объект2 {

параметр: значение;

}

На рис. 2.2 представлен пример применения красного цвета только к тегу i, который является последовательным наследником article и p. Тем самым, иерархия наследования может быть любой глубины, не только на 2 объекта.

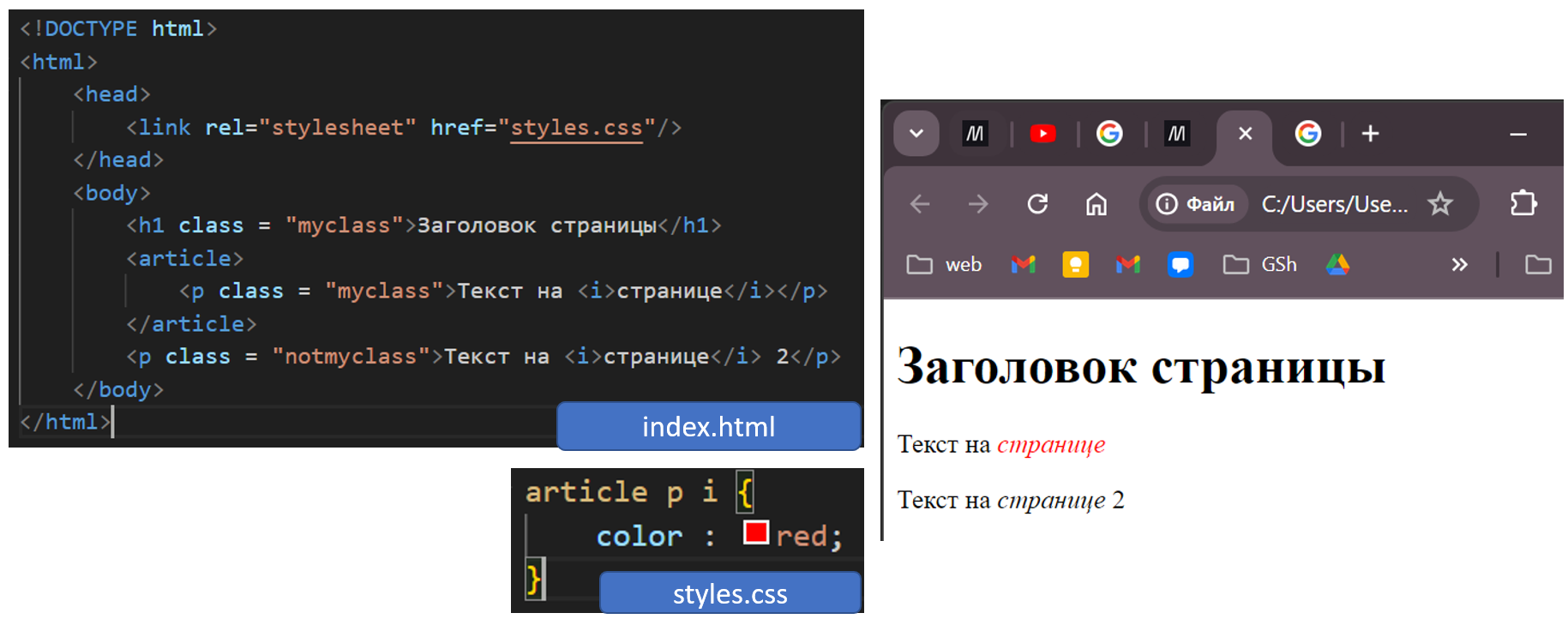


Рис. 2.2 – Пример CSS для наследника

Для объектов с несколькими состояниями можно применять правила на основе состояний, указывая их тип после «:»:

объект: состояние {

параметр: значение;

}

На рис. 2.3 представлен пример стилизации ссылки на основе состояния. В базовом состоянии link к ней применяется розовый цвет, после посещения ссылки сменяемый зеленым. В добавок к изменению цвета, при наличии курсора над ссылкой, что соответствует состоянию hover, у ссылки удаляется подчеркивание за счет правила text-decoration: none.

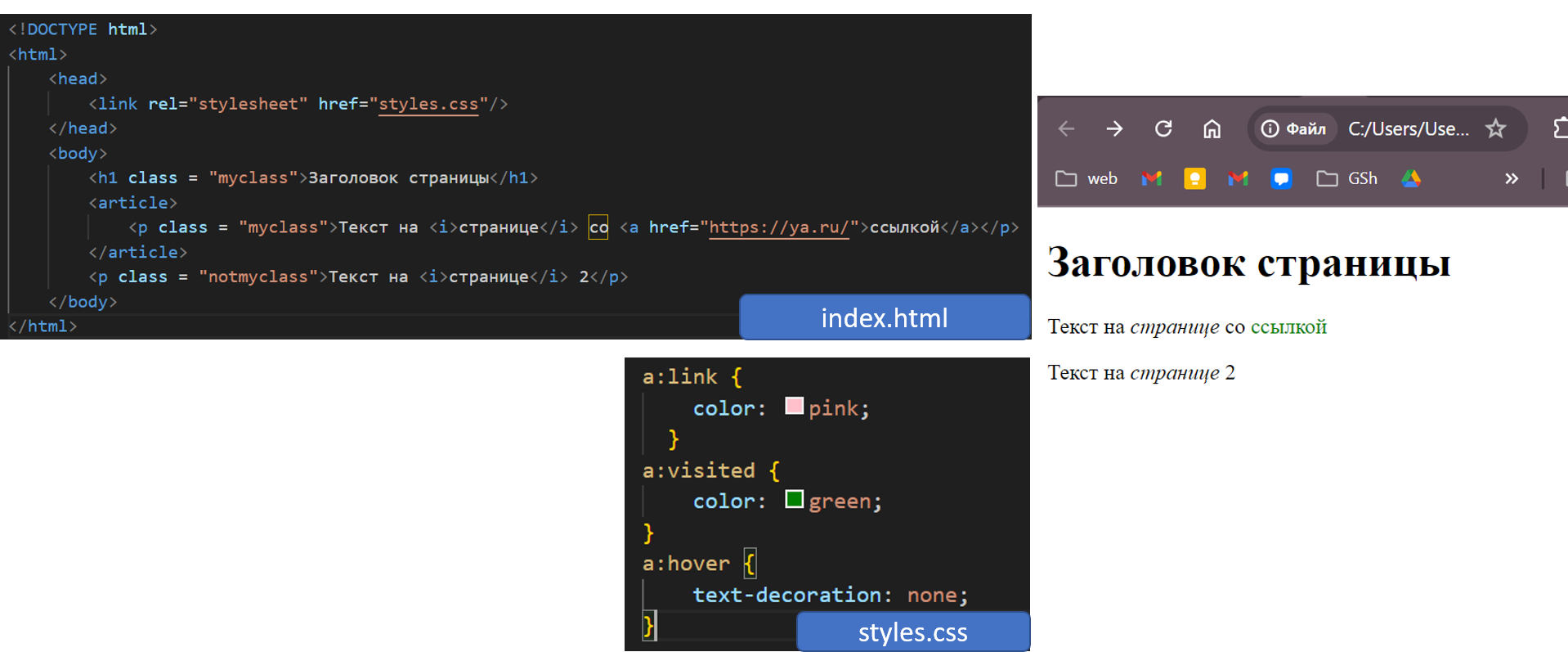


Рис. 2.3 – Пример CSS по состоянию

Ранее приведенные адаптивные изображения на самом деле тоже использовали CSS, поскольку медиа правила в первую очередь предназначена для него. И раз в CSS работают правила, можно применять какие-то параметры оформления к объекту в зависимости от ширины окна: @media позволяет задействовать правила только при выполнении условий. Ниже приведен пример, в котором правила выполняются, если окно не меньше 600px:

@media screen and (min-width: 600px){

объект {

параметр: значение;

}

}

Важно, что не следует создавать несколько правил по ширине экрана, при граничном значении правила могут быть не применены.

Запишем рассмотренные параметры CSS:

text-align – для выравнивания текста по горизонтали внутри блоков (center, end, justify, start, left, right);

margin: auto выравнивает по центру блок, если задана width<100%;

align-content – вертикальное выравнивание содержимого блока (baseline, center, end), например текста <div><p>текст</p></div>;

align-items – вертикальное выравнивание блоков внутри блока (stretch –занять все пустое пространство в flex, center, start, end);

justify-content – выравнивание блоков вдоль главной оси (горизонтальной по умолчанию);

display – меняет тип элемента (block – блочный, можно применять отступы и легче задавать положение на странице), inline – строковый (как текст), flex – растяжимые блоки; block-inline – блоки идут в строку, переносятся при сужении окна, none – не отображать;

margin – расстояние от границы блока до других объектов в %;

padding – расстояние от контента до границы блока в %;

border – параметры границы блока (цвет и тип – black solid);

background-color – цвет фона блока;

width, height – ширина и высота.

Важным в оформлении страниц является возможность добавления блокам фона. Для этого могут быть полезны два свойства CSS. Фоновое изображение блока (здесь сверху файл1, затем файл2, под ними градиент от цвета1 к цвету2):

background-image: url("файл1.jpg"),

url("файл2.jpg"),

linear-gradient(цвет1, цвет2);

Замостить фон: background-repeat: тип. Типы: repeat-x (повторить по оси x), repeat-y (по y), repeat(x+y), space (x+y с отступами), no-repeat (без повтора). Здесь приведены не все типы, но лишь основные. По умолчанию выставляется repeat, если указано фоновое изображение background-image, но background-repeat не используется. Тип space удобной использовать для формирования узора на фоне. Чтобы под изображением был виден задний фон, как здесь градиент, нужно использовать изображение, поддерживающее прозрачность, avif, png, webp, svg, но не jpg.

Чтобы изображение пропорционально меняло размер, нужно задать ему в CSS max-width: 100% и height: auto. Если изображение задается в теге picture, применять эти свойства нужно к img тегу.

Для фиксации блока при прокрутке position: fixed. Чтобы зафиксировать блок header в верхней части страницы, нужно убрать отступы по краям top: 0; right: 0; left: 0. Рекомендуется повышать приоритет строки меню z-index: 1030*, это значение стандартно для фиксированного меню, другие стандартные значения можете найти в онлайн документации сами.* Для добавления тени box-shadow: 0px -10px 10px 10px #000; (параметр: сдвиг по горизонтали; по вертикали, положительный=вверх; размытие; размер тени; цвет). Такое меню должно быть по ширине окна, width: 100%.

Если контент страницы задан в виде inline-block для адаптации к размеру окна, следует установить body position: relative для его авто масштабирования. Также следует выставить нулевые margin, height 100vh (100% высоты окна по вертикали). Чтобы footer всегда находился внизу страницы, ему нужно задать position: absolute, координаты bottom: 0; left: 0 и ширину 100% (см. рис. *2.4*).

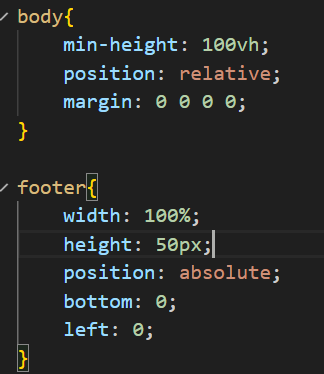


Рис. 2.4 – Absolute footer

Поскольку fixed располагается над контентом, чтобы исключить наложение рекомендуется добавить пустой div после fixed с той же высотой. Также следует поступать с absolute footer (см. рис. *2.5*).



Рис. 2.5 – Невидимые блоки

Зафиксируем параметры для работы с текстом:

font-size – размер шрифта;

font-weight – толщина букв, font-style = italic, bold, …;

font – тип шрифта, может загружаться по ссылке через link в head;

font-family – семейство шрифтов (несколько через запятую, выбирается первый из поддерживаемых).

Стандартные шрифты: serif, sans-serif, monospace, cursive, fantasy. Пример – font: fantasy.

Лабораторная работа №3. JavaScript

Подготовка к работе

Для данной работы не требуется дополнительных файлов и прочих инструментов. Используйте VS Code для работы (далее инструкции для нее).

Задания для самостоятельного выполнения

1. Создайте основу проекта и изучите работу с окнами браузера с помощью JavaScript.

1.1. Создайте папку website на рабочем столе. Откройте данную папку, как рабочее пространство (workspace, вкладка  на боковой панели – кнопка Open folder). В ней создайте текстовый документ и сохраните его с именем index.html (при сохранении выбирайте соответствующее расширение). Теперь скопируйте в созданный файл базовую структуру HTML из табл. 3.1.

Таблица 3.1 – Базовый код HTML

|  |
| --- |
| Исходный код HTML |
| <!doctype html>  <html>  <head>  <meta charset="utf-8" />  <title>Моя тестовая страница</title>  </head>  <body>  </body>  </html> |

1.2. JavaScript позволяет создавать программный код, выполняемый прямо во время загрузки страницы. Начнем с простого – открытие нового окна по щелчку левой кнопкой мыши по ссылке. Скопируйте себе внутрь body код из табл. 3.2. Здесь:

* onClick – метод, позволяющий определить, что на объект щелкнули мышью;
* window.open – метод, открывающий новую вкладку или окно, причем у него есть несколько аргументов;
* первый аргумент window.open – ссылка на сайт ('http://www.samgtu.ru'),
* второй – имя окна в программе JavaScript ('SamgtuWindow');
* положение и размер окна ('left=100, top=100, width=400, height=400'), причем, если не указать положение окна (left=100,top=100) или его ширину (width=400), высоту (height=400'), то оно откроется как новая вкладка.

1.3. После копирования сохраните документ в VS Code (Ctrl+S), откройте его же в браузере (из папки website) и проверьте результат.

Таблица 3.2 – Код открытия нового окна

|  |
| --- |
| Исходный код HTML + JavaScript |
| <a href=""  onClick="window.open('http://www.samgtu.ru',  'SamgtuWindow','left=100,top=100,width=400,height=400')">  Открыть новое окно  </a> |

Причем не обязательно использовать тег ссылки a, чтобы щелчок по нему вызывал открытие нового окна. Например, в табл. 3.3 то же действие происходит по щелчку на параграф текста, скопируйте себе данный код в body.

1.3. Скопируйте себе в body код из табл. 3.3, сохраните документ в VS Code (Ctrl+S), обновите страницу в браузере (кнопкой обновить или F5 на клавиатуре) и проверьте результат.

Таблица 3.3 – Код открытия нового окна по щелчку на параграфе текста

|  |
| --- |
| Исходный код HTML + JavaScript |
| <p  onClick="window.open('http://www.samgtu.ru','SamgtuWindow',  'left=100,top=100,width=400,height=400')">  Параграф текста  </p> |

Но нажатие на текст не слишком очевидно, поэтому следует в качестве активных элементов использовать ссылки, кнопки и подобные блоки.

1.4. Добавьте в код блок кнопки:

<input type="button" value="Моя кнопка"/>

и самостоятельно добавьте ей возможность открытия ссылки на сайт СамГТУ.

1.5. Найдите в поиске страницу погоды Яндекс для вашего города или района города, откройте сданную страницу и скопируйте ссылку на нее.

1.6. Смените ссылку в onClick кнопки на скопированную.

1.7. Затем измените имя окна на WeatherWindow.

1.8. Установите параметры положения (top, left) на значение N+10, а ширины и высоты окна на значение 2\*N+100, где N – ваш вариант по списку.

1.9. После копирования сохраните документ в VS Code (Ctrl+S), обновите страницу в браузере (кнопкой обновить или F5 на клавиатуре) и проверьте результат.

Другим простым методом, управляющим окнами, является window.close(), при его вызове текущее окно закрывается.

1.10. Добавьте в body еще одну кнопку, в ней добавьте к атрибутам onClick="" и вместо window.open() укажите window.close() внутри кавычек.

1.11. Переименуйте кнопку (атрибут value) с «Моя кнопка» на «Закрыть страницу». После копирования сохраните документ в VS Code (Ctrl+S), обновите страницу в браузере (кнопкой обновить или F5 на клавиатуре) и проверьте результат. По нажатию на кнопку страница закроется, откройте ее повторно в браузере из папки website.

2. Освойте основы проверки форм в JavaScript.

Если скрипт достаточно большой, рекомендуется подключать его в теге <script>.

2.1. Добавьте в конце body парный тег:

<script>

</script>

после чего скопируйте в него код из табл. 3.4. Данный код будет проверять верность введения данных в текстовые поля, о которых далее.

Таблица 3.4 – Код проверки формы

|  |
| --- |
| Исходный код JavaScript |
| function CheckForm(UserForm) {  if (UserForm.username.value == ''){  alert("Введите ваше имя");  UserForm.username.focus();  }  if (UserForm.email.value == ''){  alert("Введите ваш e-mail");  UserForm.email.focus();  }  if (UserForm.Comment.value == ''){  alert("Введите ваш коммениарий");  UserForm.Comment.focus();  }  } |

Здесь происходит проверка состояния форм. Рассмотрим строку первой проверки if (UserForm.username.value == ' '){}:

if() {} – проверяет, выполняется ли условие, заданное в скобках и если да, то будет выполнен код, который находится в фигурных скобках;

UserForm.username.value – узнаем, чему у формы, для которой вызывается скрипт, равно значение у тега, атрибут name которого равен username;

== – сравни, равны ли между собой;

' ' – пустая строка.

Таким образом, if (UserForm.username.value == ' '){}означает, что если в элементе формы username ничего не введено (пустая строка), то выполнится код, указанный в фигурных скобках.

Код, который выполняется, если то или иное поле формы пусто, примерно идентичен:

alert("Введите ваше имя");

UserForm.username.focus();

где первая строка выводит предупреждение, а вторая пытается вернуть курсор в то поле, которое не было заполнено.

Для того, чтобы пользователь мог вводить информацию, которую вы могли бы обрабатывать на своей странице или на сервере (например, имя, почта, комментарий), в HTML существует понятие форм – контейнера, в который можно поместить несколько полей для ввода информации пользователем и как-то отреагировать на ввод этой информации. Вы уже сталкивались с полями ввода ранее, это объекты input, частным случаем которых является кнопка. Также можно использовать для ввода input type="text" для ввода однострочных данных или <textarea rows="n"> для ввода многострочного текста (из n строк).

2.2. Скопируйте себе в body, перед <script> код из табл. 3.5, который представляет собой простую форму.

Таблица 3.5 – Код формы

|  |
| --- |
| Исходный код JavaScript |
| <form onSubmit = "return CheckForm(this)">  <input type="text" name="username"><br>  <input type="text" name="email"><br>  <textarea rows="n" name="Comment"></textarea><br>  <input type="submit" value="Отправить"><br>  </form> |

У формы есть метод onSubmit, позволяющий вызывать какой-то скрипт по заполнению формы. Само заполнение формы проверяется нажатием кнопки, которая задается как <input type="submit"> (submit вместо button).

2.3. Замените в данной форме, в теге textarea число строк с n на N+5, где N – номер вашего варианта.

2.4. Сохраните документ в VS Code (Ctrl+S), обновите страницу в браузере (кнопкой обновить или F5 на клавиатуре) и проверьте результат – должна появиться форма, при отправке которой, если какое-то поле осталось пустым, выводится сообщение о том, что оно не заполнено.

2.5. Продублируйте все условия в JavaScript коде (выделите, скопируйте и вставьте там же в новую пустую строку. Для полученных копий измените пустую строку (' '), с которой сравнивается значение поля, на какое-то, которое будет являться верным (например, 'Вася'). Удалите в каждой копии строку с фокусом на поле и исправьте выводимые сообщения на позитивные (например,   
'Молодец! Введено лучшее имя!' и т.д.). После этого сохраните документ в VS Code (Ctrl+S), обновите страницу в браузере (кнопкой обновить или F5 на клавиатуре) и проверьте результат.

3. Динамическое изменение стилей.

JavaScript позволяет не только выполнять действия вроде открытия новых окон, но и динамически изменять содержимое уже загруженной страницы.

Для начала научимся изменять HTML-элементы по нажатию на кнопку. Чтобы JavaScript мог найти нужный элемент на странице, этому элементу необходимо задать уникальный идентификатор с помощью атрибута id.

3.1. Добавьте в body перед тегом <script> код из табл. 3.6.

Таблица 3.6 – Код для изменения стиля элемента

|  |
| --- |
| Исходный код JavaScript |
| <p id="text\_to\_change">Этот текст изменит свой цвет.</p><br>  <input type="button" value="Изменить цвет"  onClick="document.getElementById('text\_to\_change')  .style.color='red'"> |

Здесь мы добавили параграф с id="text\_to\_change". Кнопка ниже по нажатию (onClick) выполняет следующий код:

* document.getElementById('text\_to\_change') – этот метод находит на странице элемент по его уникальному id.
* .style.color='red' – это свойство позволяет изменить CSS-стиль найденного элемента, в данном случае – цвет текста (color) на красный (red).

3.2. Сохраните документ в VS Code (Ctrl+S), обновите страницу в браузере (кнопкой «обновить» или F5 на клавиатуре) и проверьте результат.

Теперь самостоятельно измените код так, чтобы цвет текста менялся на синий (blue).

Другим частым действием является скрытие или отображение элемента. Это контролируется через CSS-свойство display. Если display равен 'none', элемент скрыт. Если display равен 'block' или другому соответствующему значению, элемент виден.

3.3. Скопируйте в body код из табл. 3.7.

Таблица 3.7 – Код для скрытия и отображения элемента

|  |
| --- |
| Исходный код HTML+JavaScript |
| <p id="toggle\_element">Этот элемент можно скрыть или показать.</p><br>  <input type="button" value="Скрыть"  onClick="document.getElementById('toggle\_element')  .style.display='none'"><br>  <input type="button" value="Показать"  onClick="document.getElementById('toggle\_element')  .style.display='block'"> |

3.4. Cохраните документ в VS Code (Ctrl+S), обновите страницу в браузере и проверьте, как работают кнопки «Скрыть» и «Показать».

3.5. Создайте внутри тега <script> еще одну функцию с именем getRandomColor.

3.6. В функции нужно реализовать формирование строки со случайным цветом, для этого необходимо использовать:

* Math.floor(Math.random() \* 10); для вычисления случайного значения от 0 до 9 (Math.random() дает float от 0 до 1, Math.floor округляет вниз до целого);
* сложение строки '#' с 6 случайными числами (в JavaScript сложение чисел со строкой вызывает преобразование чисел в строки);
* возврат с помощью return полученной строки.

При этом для всех сущностей функции достаточно использовать константы (const) или наполнять строку, изначально равную '#' в цикле for:

for (let i = 0; i < максимальноеЗначение; i++) {тело цикла};

и вернуть уже готовую строку.

3.7. Cохраните документ в VS Code (Ctrl+S), обновите страницу в браузере и проверьте результат.

3.8. Сейчас цвета ограничены неполным диапазоном из-за отсутствия при генерации цвета символов A,B,C,D,E,F (при задании цвета в hex-формате они дают наибольшие значения). Измените функцию getRandomColor так, чтобы с некоторой вероятностью вместо прибавления к строке числа прибавлялась строка со случайно выбранной из A,B,C,D,E,F буквой.

3.9. Cохраните документ в VS Code (Ctrl+S), обновите страницу в браузере и проверьте результат.

4. Работа со временем.

JavaScript позволяет выполнять код с заданной периодичностью. Этим можно воспользоваться для создания часов, которые будут обновляться в реальном времени без перезагрузки страницы.

Для этого используются:

* new Date() – объект, который содержит текущую дату и время.
* setInterval(function, milliseconds) – метод, который выполняет указанную функцию каждые milliseconds миллисекунд (1000 миллисекунд = 1 секунда).

4.1. Добавьте в body (например, после кнопок) код из табл. 3.8.

Таблица 3.8 – Код для отображения текущего времени

|  |
| --- |
| Исходный код HTML+JavaScript |
| <p>Текущее время:</p>  <span id="clock"></span> <br>  <script>  function updateTime() {  let today = new Date();  document.getElementById('clock').innerHTML=today.toLocaleTimeString();  }  setInterval(updateTime, 1000);  </script> |

Рассмотрим код в теге <script>:

* сначала создается пустой тег <span id="clock"></span>, в который будем выводить время;
* function updateTime() – объявляем функцию, которая будет обновлять время;
* let today = new Date(); – получаем текущие дату и время.
* document.getElementById('clock').innerHTML = today.toLocaleTimeString(); – находим span по id и помещаем внутрь него (innerHTML) время, преобразованное в строку с помощью метода toLocaleTimeString();
* setInterval(updateTime, 1000); – указываем браузеру запускать функцию updateTime каждую секунду (1000 мс).

4.2. Сохраните документ (Ctrl+S), обновите страницу в браузере и проверьте результат. Вы должны увидеть время, которое обновляется каждую секунду.

4.3. Измените код так, чтобы интервал обновления был равен 1000 + N\*50 миллисекунд, где N – ваш номер по списку. Проверьте результат.

5. Реализация калькулятора.

Создадим простую форму, которая будет складывать два числа. Для этого нам понадобятся два поля для ввода чисел, кнопка для запуска вычисления и место для вывода результата.

5.1. Скопируйте в body перед основным тегом <script> код из таблицы 3.9.

Таблица 3.9 – Код для выполнения арифметических операций

|  |
| --- |
| Исходный код HTML+JavaScript |
| <h3>Простой калькулятор</h3>  Число 1: <input type="text" id="num1"><br>  Число 2: <input type="text" id="num2"><br>  <input type="button" value="Сложить" onClick="calculate()">  Результат: <span id="result"></span>  <script>  function calculate() {  let n1 = document.getElementById('num1').value;  let n2 = document.getElementById('num2').value;  let sum = parseInt(n1) + parseInt(n2);  document.getElementById('result').innerHTML = sum;  }  </script> |

В функции calculate():

* let n1 = document.getElementById('num1').value; – получаем значение (value), введенное пользователем в поле с id="num1". Важно помнить, что все, что вводится в поле input type="text", является строкой. Чтобы JavaScript рассматривал это как число, используется функцию parseInt(), которая преобразует строку в целое число;
* let sum = parseInt(n1) + parseInt(n2); – мы складываем два преобразованных числа;
* document.getElementById('result').innerHTML = sum; – выводим полученную сумму в span с id="result".

5.2. Сохраните документ (Ctrl+S), обновите страницу, введите два числа в поля и нажмите кнопку «Сложить» для проверки.

5.3. Добавьте новое поле с текстом «Операнд» после поля «Число 2»;

5.4. Измените код функции calculate() так, чтобы она применяла между заданными числами введенный операнд (возможные варианты: /, \*, +, -, %).

5.5. С помощью конструкций условных операторов if-else реализуйте возможность применения операции, соответствующей переданному в поле «Операнд» символу между введенными числами.

5.6. Измените название кнопки на «Вычислить».

После этого самостоятельно добавьте еще одну кнопку «Вычесть», которая будет вычитать второе число из первого. Для этого вам потребуется создать новую функцию (например, subtract()) по аналогии с calculate() и привязать ее к onClick новой кнопки.

6. Работа с асинхронностью.

Используя асинхронное программирование, создайте функцию, которая будет имитировать загрузку данных с задержкой, а затем отображать их на странице.

6.1. Добавьте в <head> вашего файла index.html элемент <style> и внутри него напишите CSS-правила:

* для визуального отображения процесса загрузки, создайте класс .loading, который будет делать текст синим и полужирным с помощью атрибутов color и font-weight;
* для индикации успешной загрузки добавьте класс .success (с зеленым цветом), а для ошибки – класс .error (с красным цветом);

6.2. В теле страницы <body> добавьте заголовок <h3>Асинхронная загрузка данных</h3> и под ним пустой элемент <span> с идентификатором id="data-status". Присвойте ему класс class="loading" и поместите внутрь текст «Загрузка...». Это начальное состояние, которое увидит пользователь.

6.3. В теге <script> создайте асинхронную функцию с именем fetchDataWithDelay. Эта функция будет возвращать промис – специальный объект, который представляет результат асинхронной операции в будущем.

6.4. Создание промиса. Внутри функции создайте новый промис с помощью new Promise(). Он принимает в качестве аргумента функцию-исполнитель (executor), которая имеет два параметра: resolve (для успешного завершения) и reject (для ошибки). Пример такого кода приведен в табл. 3.10.

Таблица 3.10 – Код для промиса

|  |
| --- |
| Исходный код JavaScript |
| function fetchDataWithDelay() {  return new Promise((resolve, reject) => {  // ... код имитации задержки  });  } |

6.5. Чтобы сымитировать время, необходимое для загрузки данных, используйте функцию setTimeout(). Она позволяет отложить выполнение кода на определенное время (см. табл. 3.11).

Таблица 3.11 – Код имитации задержки

|  |
| --- |
| Исходный код JavaScript |
| setTimeout(() => {  // Этот код выполнится через 3000 миллисекунд (3 секунды)  }, 3000); |

6.6. Внутри setTimeout вызовите функцию resolve(), передав ей результат, который вы хотите вернуть. В данном случае, это строка: «Данные успешно загружены! (за 3 секунды)». Вызов resolve переведет промис в состояние fulfilled (выполнено), и его результат будет доступен.

6.7. После определения функции fetchDataWithDelay, вызовите ее. Примените к ней метод .then() для обработки успешного результата. Метод .then() принимает функцию, которая будет выполнена, когда промис перейдет в состояние fulfilled (вызван resolve).

6.8. Сам .then() нужно оформить в виде стрелочной функции. Задайте ей имя аргумента (например, data), вместо него в теле метода будет подставлено собранное resolve() значение.

Пример стрелочной функции:

имя\_функции(аргумент => (тело функции));

в таком виде вам нужно задать вызов .then сразу после вызова fetchDataWithDelay. Лучше вызвать данный метод с новой строки, например:

функция().

.метод();

6.9. Найдите использованным ранее кодом элемент HTML с id="data-status". Измените свойство innerHTML данного элемента на строку, полученную из промиса.

6.10. Затем измените атрибут в том же методе className на 'success', чтобы применить стили для успешного результата. Этот атрибут меняется также, как innerHTML.

6.11. Добавьте метод .catch() в цепочку промисов сразу после .then(). Метод .catch() предназначен для обработки ошибок. Он сработает, если промис будет отклонен (вызван rejected).

6.12. Внутри setTimeout, добавьте логику, которая будет случайным образом вызывать ошибку (сымитируем ее). Если случайное число будет меньше 0.1 + N/100 (где N –номер вашего варианта по списку), вызовите функцию reject(), передав ей новый объект Error (new Error('сообщение')) с сообщением: "Ошибка загрузки данных!".

6.13. Внутри .catch() найдите элемент с id="data-status", измените его содержимое на сообщение об ошибке (переменная\_с\_ошибкой.message) и присвойте ему класс 'error' для визуального отображения.

6.14. Сохраните документ и обновите страницу в браузере. Вы должны увидеть текст «Загрузка...», который через 3 секунды сменится на «Данные успешно загружены!» (если не произошла ошибка).

6.15. Обновляйте страницу 10 раз, запишите процент удачных «загрузок данных» в вашем случае.

7. Использование синтаксиса async/await.

В этом разделе вы перепишете ваш код из предыдущего задания, используя async/await. Этот подход позволяет избежать цепочек .then() и делает код более читаемым и легким для отладки.

7.1. Создайте новую кнопку с текстом «Загрузить с помощью async/await» и присвойте ей атрибут onClick="loadDataAsync()".

7.2. В теге <script> создайте новую асинхронную функцию с именем loadDataAsync. Для этого перед именем функции добавьте ключевое слово async.

7.3. Внутри функции loadDataAsync создайте блок try...catch (try{} catch(error){}). Это аналог .then() и .catch() для обработки успешного выполнения и ошибок соответственно.

* Код, который может вызвать ошибку, поместите в блок try.
* Код для обработки ошибки поместите в блок catch. В catch передайте аргумент error.

7.4. Внутри блока try используйте ключевое слово await перед вызовом вашей функции fetchDataWithDelay(). await «поставит на паузу» выполнение функции loadDataAsync до тех пор, пока промис, возвращаемый fetchDataWithDelay, не будет разрешен.

7.5. Найдите элемент id="data-status" и измените его текст на «Загружаю асинхронно...». Очистите его класс, присвоив ему пустую строку (className = ''), чтобы убрать предыдущие стили. Это действие нужно выполнить перед вызовом await, чтобы пользователь сразу увидел индикацию загрузки.

7.6. Присвойте результат выполнения промиса await fetchDataWithDelay() в переменную (например, const message). Затем, как и в предыдущем задании, найдите элемент id="data-status", измените его innerHTML на message и присвойте ему класс 'success'.

7.7. В блоке catch(error), найдите элемент id="data-status" и измените его innerHTML на error.message. Присвойте ему класс 'error' для отображения ошибки.

7.8. Сохраните документ и обновите страницу. Проверьте, что обе кнопки (.then() и async/await) работают одинаково, но при этом код с async/await выглядит более понятным и «плоским» (без вложенных блоков).

8. Задания без пояснений. В данном блоке даются задания, выполнить которые вы должны полностью самостоятельно, демонстрируя усвоение полученных в прошлых заданиях навыков.

8.1. Создайте форму из трех полей и кнопки подтверждения формы, в ней организуйте проверку данных: в первом и втором поле допускаются только числа, в третьем – только операнды, рассмотренные ранее. Проверять, является ли строка числом нужно преобразовав ее в число функцией Number, а затем используя функцию isNaN, которая возвращает true, если переданное ей значение числом не является.

8.2. Добавьте параграф текста после формы. По нажатии кнопки должна осуществляться проверка формы, и если все проверки пройдены успешно – записывать в параграф рассчитанное значение операции между двумя числами, если что-то введено некорректно, то в параграф нужно записать текст «Некорректный ввод».

8.3. Задайте новый параграф текста, который раз в секунду менял бы свой цвет на случайный.

Пояснения к выполнению

Для выполнения последних заданий может потребоваться некоторое напоминание про принципы работы промисов в JavaScript. Промис, который представляет собой объект, символизирующий будущее завершение (или неудачу) асинхронной операции.

Он может находиться в одном из трех состояний:

* pending: начальное состояние, операция еще не завершена.
* fulfilled: операция успешно завершена.
* rejected: операция завершилась с ошибкой.

Для обработки результата промиса используются методы .then() для успешного выполнения и .catch() для ошибок.

Для более удобной работы с асинхронным кодом используется синтаксис async/await. Функция, объявленная с ключевым словом async, может использовать оператор await для приостановки своего выполнения до тех пор, пока промис не будет разрешен.

Лабораторная работа №4. JavaScript и хранение данных

Подготовка к работе

Для выполнения данной работы нужно установить расширение Live Server в VS Code от Ritwick Dey. Дело в том, что в данной работе мы будем использовать скрипты, сохраненные в виде файлов, а не встроенные в код, как в прошлой работе. Но выполнение локальных файлов скриптов заблокировано из соображений безопасности в современных браузерах, поэтому придется работать в режиме локального сервера – браузеру будет казаться, что весь проект получен от сервера (адрес начинается на https), а не из файлового хранилища (адрес начинается на file). Работа предполагает создание проекта с определенной структурой файлов, которая будет описана в задании.

Задания для самостоятельного выполнения

1. Создайте структуру проекта (для лучшей организации кода проект будет разделен на несколько файлов).

1.1. Создайте на рабочем столе папку с именем notes-app.

1.2. Создайте папку website на рабочем столе. Откройте в VS Code данную папку, как рабочее пространство (workspace, вкладка  на боковой панели – кнопка Open folder). В ней создайте текстовый документ и сохраните его с именем index.html для разметки страницы (при сохранении выбирайте соответствующее расширение). Таким же образом создайте файлы: style.css для оформления, main.js для основной логики приложения и storage.js для функций, отвечающих за хранение данных.

1.3. Скопируйте в index.html базовую структуру из табл. 4.1.

Таблица 4.1 – Базовая структура HTML

|  |
| --- |
| Исходный код HTML |
| <!doctype html>  <html>  <head>  <meta charset="utf-8" />  <title>Приложение для заметок</title>  <link rel="stylesheet" href="style.css" />  </head>  <body>  <script src="main.js" type="module"></script>  </body>  </html> |

Обратите внимание на атрибут type="module" в теге <script>. Он необходим, чтобы браузер мог корректно работать с модульной системой JavaScript, позволяющей импортировать и экспортировать функции между файлами (main.js и storage.js).

2.Наполните HTML-интерфейс.

2.1. В файле index.html внутри тега <body>, перед подключением скрипта, добавьте HTML-код для интерфейса приложения из табл. 4.2.

Таблица 4.2 – HTML-код интерфейса

|  |
| --- |
| Исходный код HTML |
| <h1>Мои заметки</h1>  <form id="note-form">  <input type="text" id="note-title" placeholder="Заголовок" required> <br>  <textarea id="note-body" placeholder="Текст заметки" required></textarea>  <button type="submit">Сохранить заметку</button>  </form>  <h2>Сохраненные заметки:</h2>  <div id="notes-container"></div> |

Данный код создает заголовок, форму для ввода данных (<form>), состоящую из поля для заголовка (<input>), текстового поля для содержания заметки (<textarea>) и кнопки для отправки. Блок <div id="notes-container"> будет использоваться как контейнер для динамического отображения сохраненных заметок. Аргумент required у элементов формы указывает на необходимость заполнения этого элемента, тем самым проверка на заполненность происходит без дополнительного скрипта.

3. Реализуйте модуль работы с хранилищем (storage.js).

На этом шаге будут созданы функции для сохранения и загрузки данных из localStorage. Это локальное хранилище браузера, которое позволяет сайтам сохранять данные (в виде пар "ключ-значение") без срока истечения. Это необходимо, чтобы заметки пользователя не исчезали после перезагрузки страницы или закрытия браузера.

3.1. Откройте файл storage.js и скопируйте в него код функции saveNotes из табл. 4.3.

Таблица 4.3 – Функция сохранения заметок

|  |
| --- |
| Исходный код JavaScript |
| export function saveNotes(notes) {  const notesJSON = JSON.stringify(notes);  localStorage.setItem('app-notes', notesJSON);  } |

Ключевое слово export делает эту функцию доступной для импорта в других файлах. Функция принимает массив заметок (notes), преобразует его в строку формата JSON с помощью метода JSON.stringify() (поскольку в localStorage можно хранить только строки) и сохраняет в хранилище под ключом 'app-notes' с помощью метода localStorage.setItem().

3.2. Добавьте в тот же файл storage.js код функции getNotes из таблицы 4.4.

Таблица 4.4 – Функция получения заметок

|  |
| --- |
| Исходный код JavaScript |
| export function getNotes() {  const notesJSON = localStorage.getItem('app-notes');  try {  return notesJSON ? JSON.parse(notesJSON) : [];  } catch (e) {  return [];  }  } |

Пояснение: Эта функция извлекает данные по ключу 'app-notes' с помощью метода localStorage.getItem(). Поскольку они хранятся как строки, они преобразуются обратно в массив объектов с помощью JSON.parse(). Блок try...catch используется для обработки возможных ошибок, если данные в хранилище были повреждены. Если данных нет, функция вернет пустой массив, что предотвратит ошибки в дальнейшей работе.

4. Реализуйте основную логику приложения в main.js.

4.1. В файле main.js сначала импортируйте созданные функции. Добавьте в начало файла следующую строку:

import { getNotes, saveNotes } from './storage.js';

4.2. Создайте функцию renderNotes() для отображения заметок на странице. Эта функция будет считывать все заметки и динамически создавать для них HTML-элементы. Предположим такую структуру заметки: заголовок, текст заметки, ее идентификатор. Для удобства сохранять заметку будем в формате:

{"id":идентификатор, "title":"заголовок", "text":"текст заметки"}

и тогда, например, для получения текста заметки, сохраненной в переменной note нужно будет воспользоваться кодом note.text.

Внутри renderNotes():

* получите массив заметок с помощью getNotes(), запишите его в какую-то переменную;
* получите элемент-контейнер для заметок (document.getElementById('имя-контейнера');
* очистите его, установив свойство innerHTML в пустую строку (''), чтобы избежать дублирования заметок при повторном вызове функции;
* используйте цикл (forEach или for...of), чтобы пройтись по каждой заметке в массиве:

array.forEach( (currentValue, index, arr) => {

// код, который будет выполнен для каждого элемента

});

где currentValue – текущий элемент в массиве, обязательный аргумент;

index – индекс текущего элемента, не обязательный;

arr – весь итерируемый массив, не обязательный.

Таким образом forEach аналогичен простому for item in array в Python. Если перебирать только элементы массива, то имя аргумента стрелочной функции в скобки можно не брать, например:

my\_arr.forEach( item => {

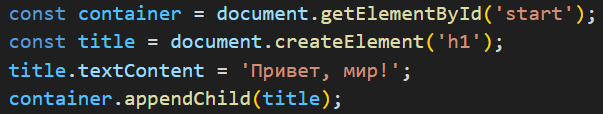
console.log(item);

}

будет аналогичен for item in my\_array: print(item) на Python;

* для каждой заметки создайте HTML-разметку, которая будет включать заголовок (<h3>), текст заметки (<p>) и кнопку «Удалить»;
* в заголовок запишите атрибут title переменной для текущей заметки в цикле (значение.title), в параграф – атрибут text;
* Добавьте кнопке data-атрибут, который будет содержать номер заметки по порядку в массиве из readNotes(). Это позволит позже понять, какую именно заметку нужно удалить. Задайте data-атрибут id и присвойте ему значение атрибута id заметки.

Стоит напомнить, что для создания нового элемента на странице используется createElement, а для размещения созданного – appendChild, применяемый для объекта, после которого добавляется новый объект с самим новым объектом в качестве аргумента. Чтобы добавить объект после конкретного блока, нужно выбрать этот блок с помощью getElementById и для него вызвать метод appendChild. Код на рис. 4.1 создаст после блока с id start заголовок h1 с текстом «Привет, мир!».



*Рис. 4.1.* Пример создания объекта HTML в коде JavaScript

В HTML5 появились атрибуты, которые позволяют хранить произвольные данные прямо в элементе HTML-разметки. Его название всегда начинается с префикса «data-». Для его присвоения в JavaScript объекту используется конструкция объект.dataset.атрибут1 = 'данные' (так у объекта появится свойство data-атрибут1 со значением 'данные').

4.3. В конце файла main.js вызовите renderNotes() один раз, чтобы при загрузке страницы отобразились все ранее сохраненные заметки.

4.4. Для проверки работы страницы здесь и далее используйте LiveServer – одноименную кнопку в правом нижнем углу VS Code при открытом в нем файле index.html (если ее нет, ищите обозначение запрещающего знака и слово Port в том же месте, нажимайте на него). Для отладки кода можно добавить в него строки console.log('текст для отладки') и открыть в браузере консоль разработчика (например, в Chrome нажать правой кнопкой мыши по странице и выбрать «Посмотреть код»). В данном меню найдите раздел консоль (Console), здесь будут отображаться ошибки (не только в JavaScript) и/или ваши сообщения console.log.

4.5. Проверьте работоспособность функции, для чего закомментируйте строку в renderNotes, ответственную за загрузку сохраненных заметок. Под этой строкой добавьте другую с тем же именем переменной и содержимым:

[{"id":0, "title":"один", "text":"первый"}, {"id":1, "title":"два", "text":"второй"}]

которое подобно массиву в JSON-формате (является словарем, как в Python). Таким образом, каждый объект заметки – это JSON с тремя полями (id, title, text) и при создании нового объекта нужно формиро

Теперь при загрузке страницы должны выводиться две заметки с заданным вами содержимым. Если этого не происходит – ищите причину ошибки;

* смените индекс цикла forEach в присвоении data-атрибута id к кнопке на атрибут id от заметки.

4.6. Добавьте обработчик события submit на форму с идентификатором note-form:

* в отличие от прошлой работы, назначим обработчик в самом коде JavaScript, без атрибута onclick у кнопки. Для этого нужно создать константу и получить по id объект кнопки в нее (getElementById), после чего назначить этой переменной обработчик нажатия (submit по сути тоже нажатие) с помощью

переменная\_кнопки.addEventListener('submit', обработчик);

где обработчик – имя функции, которую нужно выполнить по нажатию на кнопку подтверждения формы (для обычной кнопки, не относящейся к подтверждению формы, вместо submit нужно использовать click);

* внутри обработчика отмените стандартное поведение отправки формы с помощью добавления event.preventDefault() в качестве первой строки в теле функции обработчика. Также, несмотря на то, что event глобальная переменная по умолчанию, считается устаревшим такое использование preventDefault. Современный подход – передавать event в качестве аргумента в функцию-обработчик, сделайте в своем обработчике также;
* далее в обработчике запишите в константы значения из полей ввода (note-title и note-body). Для получения значения объекта, найденного в HTML, нужно считать его атрибут value (объект.value);
* создайте новый объект заметки с уникальным id (например, на основе текущего времени Date.now(), который вернет число секунд, прошедших с 1 января 1970 г), а также с title и body.
* получите существующий массив заметок с помощью getNotes(), добавьте ему в конец новую заметку (массив.push(значение), аналог append в Python, ничего не возвращает, но добавляет значение в конец массива) и вызовите saveNotes() для сохранения обновленного массива;
* вызовите renderNotes() для обновления интерфейса;
* в renderNotes() раскомментируйте строку для вызова списка заметок и удалите заданный ранее массив данных двух заметок. Проверьте результат. Поскольку saveNotes() использует localStorage, заметки будут сохраняться в кэше браузера, пока вы его не очистите, а значит после закрытия и нового открытия страницы заметки не будут утеряны.

4.7. Реализуйте удаление заметки:

* создайте обработчик по клику (click) на объект с id notes-container, отмените ему действие по умолчанию;
* внутри обработчика проверьте, был ли клик совершен по кнопке удаления, например, по классу 'delete-btn':

if (event.target.matches('.имя-класса)) {

код, если нажали на объект класса имя-класса }

* если да, получите id заметки из ее data-атрибута (event.target.dataset.имя\_data\_атрибута);
* получите текущий массив заметок и используйте метод filter(), чтобы создать новый массив, исключив из него заметку с этим id:

массив\_заметок.filter(note => note.id !== исключаемый\_id);

данный метод вернет массив без объекта с исключаемым id;

* сохраните новый массив и вызовите renderNotes() для обновления списка заметок на странице;
* в renderNotes() присвойте кнопке то же имя класса, что при проверке использовалось в обработчике (переменная\_кнопки.className = 'имя-класса').

Здесь использовано делегирование событий: один обработчик события click использован для всего контейнера notes-container. При срабатывании события мы проверили, какой конкретно элемент вызвал событие внутри контейнера (if (event.target.matches('.имя-класса))) и получив id кнопки поняли, с каким объектом в контейнере нужно работать. Это очень удобное решение, если в контейнере множество объектов и создавать для каждого из них свой обработчик не имеет смысла.

5. Сравнение localStorage и sessionStorage

sessionStorage – это еще один вид хранилища, похожего на localStorage, но данные в нем существуют только в рамках одной сессии (т.е., пока открыта вкладка браузера).

5.1. Добавьте в index.html, после контейнера для заметок, следующий код:

<p>Счетчик обновлений сессии: <span id="session-counter">0</span></p>

5.2. В main.js, вне каких-либо функций, добавьте код для работы со счетчиком.

Таблица 4.5 – Код работы со счетчиком

|  |
| --- |
| Исходный код JavaScript |
| let reloadCount = sessionStorage.getItem('page-reloads') || 0; reloadCount++; sessionStorage.setItem('page-reloads', reloadCount); document.getElementById('session-counter').textContent = reloadCount; |

5.3. Сохраните все файлы, откройте index.html в браузере. Проверьте работу приложения: добавьте несколько заметок, перезагрузите страницу (заметки должны остаться, а счетчик сессии увеличиться). Затем закройте вкладку и откройте файл заново (заметки останутся, а счетчик сессии сбросится на 1).

5.4. Сохраните результаты, полученные файлы понадобятся вам для выполнения следующей лабораторной работы. Например, можете сохранить все рабочее пространство в файл (File – Save workspace as…). После сдачи работы вы сможете закрыть рабочее пространство (File – Close workspace) за ненадобностью до следующего занятия, а на нем открыть файл рабочего пространства, восстановится иерархия файлов в Explorer и вкладки проекта.

Пояснения к выполнению

Для сохранения информации на стороне клиента, чтобы она была доступна между сеансами, в современных браузерах существует Web Storage API. Этот механизм предоставляет два объекта для хранения данных в виде пар "ключ-значение": localStorage и sessionStorage. Это более современная и удобная альтернатива использованию cookie-файлов для хранения данных приложения.

localStorage представляет собой постоянное хранилище. Данные, записанные в него, не имеют срока действия и сохраняются даже после закрытия и повторного открытия браузера. Они доступны для всех окон и вкладок, открытых с одного и того же источника (домена). Это делает localStorage идеальным инструментом для хранения настроек пользователя, состояния приложения или, как в данной работе, списка заметок. Для управления данными используются методы localStorage.setItem('key', 'value') для записи, localStorage.getItem('key') для чтения и localStorage.removeItem('key') для удаления данных по ключу.

Важной особенностью Web Storage является то, что он может хранить только строковые значения. Если попытаться сохранить сложную структуру данных, такую как объект или массив, она будет автоматически преобразована в строку (например, [object Object]), что приведет к потере данных. Для корректного сохранения сложных структур их необходимо предварительно сериализовать, то есть преобразовать в строковое представление. Стандартным форматом для этого является JSON (JavaScript Object Notation). Для преобразования объекта или массива в строку JSON используется метод JSON.stringify(). Для обратного преобразования, или десериализации, строки JSON в объект JavaScript используется метод JSON.parse().

sessionStorage работает практически идентично localStorage и предоставляет тот же набор методов для управления данными. Ключевое отличие заключается в жизненном цикле хранимых данных. Хранилище sessionStorage привязано к конкретной сессии браузера, то есть к одной вкладке. Как только пользователь закрывает вкладку или браузер, все данные, сохраненные в sessionStorage для этой сессии, безвозвратно удаляются. Это делает его удобным для хранения временной информации, которая актуальна только во время одного сеанса работы пользователя с сайтом, например, данные, введенные в многошаговую форму, или состояние временного фильтра.

По мере усложнения веб-приложений объем кода на JavaScript может значительно вырасти. Хранение всего кода в одном большом файле затрудняет его чтение, отладку и поддержку. Для решения этой проблемы в современном JavaScript используется модульная система (ES6 Modules), которая позволяет разбивать код на логические части – модули. Каждый файл представляет собой отдельный модуль. Такой подход способствует лучшей организации проекта и позволяет повторно использовать код.

Для управления зависимостями между модулями используются ключевые слова export и import. Инструкция export используется в модуле, чтобы сделать его функции, переменные или классы доступными для использования в других частях приложения. Инструкция import, в свою очередь, позволяет другому модулю подключить и использовать экспортированную функциональность.

Чтобы браузер мог корректно обрабатывать модульную структуру и распознавать синтаксис import/export, необходимо при подключении основного файла скрипта в HTML-документе указать атрибут type="module". Например: <script src="main.js" type="module">. Скрипты, подключенные таким образом, также выполняются в строгом режиме (strict mode) и имеют отложенное исполнение (аналогично атрибуту defer), что предотвращает блокировку отрисовки страницы.

Лабораторная работа №5. Современный JavaScript и работа с API

Подготовка к работе

Для выполнения данной работы вам потребуется среда разработки VS Code с установленным расширением Live Server. Работа предполагает использование структуры проекта из лабораторной работы №4, включающей файлы index.html, style.css и main.js. Сперва откройте свой проект.

В данной работе будет использоваться сторонняя библиотека Chart.js для построения графиков. Для ее подключения добавьте следующую строку в тег <head> вашего файла index.html:

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

Это пример подключения библиотеки через **CDN** (Content Delivery Network), что избавляет от необходимости скачивать файлы локально.

Задания для самостоятельного выполнения

1. Получение и отображение данных с помощью **fetch** и **async/await**

На этом шаге вы напишете код для получения данных из публичного API и их отображения на веб-странице, используя современные возможности JavaScript. Для корректной обработки асинхронной операции получения данных с сервера будет использоваться синтаксис async/await.

1.1. В файле index.html внутри тега <body> добавьте HTML-код для интерфейса.

Таблица 5.1 – HTML-код интерфейса

|  |
| --- |
| Исходный код HTML |
| <h1>Курсы валют</h1>  <p>Дата последнего обновления:  <span id="date-span"></span></p>  <button id="load-rates-btn">Загрузить курсы</button>  <div id="rates-container"></div> |

1.2. В main.js получите ссылки на созданные HTML-элементы с помощью document.getElementById().

Таблица 5.2 – Код для получения элементов

|  |
| --- |
| Исходный код JavaScript |
| const loadBtn = document.getElementById('load-rates-btn');  const ratesContainer = document.getElementById('rates-container');  const dateSpan = document.getElementById('date-span');| |

1.3. Создайте асинхронную функцию fetchAndRenderRates. В ней будет происходить запрос данных и вызов функции для их отрисовки. Для отправки запроса используется функция fetch(), которая возвращает промис. Важно проверять свойство ok у ответа, так как fetch не отклоняет промис при HTTP-ошибках, таких как 404 или 500.

Таблица 5.3 – Асинхронная функция

|  |
| --- |
| Исходный код JavaScript |
| async function fetchAndRenderRates() {  // В качестве примера используем бесплатный API  const url = 'https://www.cbr-xml-daily.ru/daily\_json.js';  try {  const response = await fetch(url);  if (!response.ok) {  throw new Error('Сетевая ошибка: ' + response.status);  }  const data = await response.json();  renderRates(data);  } catch (error) {  console.error('Ошибка при загрузке данных:', error);  ratesContainer.textContent = 'Не удалось загрузить данные.';  } } |

1.4. Теперь создайте функцию renderRates(data), которая будет отвечать за отображение полученных данных. В этой функции вы примените **деструктурирующее присваивание** для удобного извлечения данных из объекта.

1.5. Внутри renderRates(data):

* извлеките свойства Date и Valute из объекта data в отдельные константы;
* отобразите дату обновления в dateSpan;
* очистите ratesContainer;
* с помощью цикла for...in переберите все дочерние свойства объекта Valute.
* Для каждой валюты создайте div. Для формирования его содержимого используйте шаблонные строки, которые позволяют встраивать переменные и выражения прямо в строку с помощью конструкции ${...}.
* Добавьте созданный элемент в ratesContainer.

Таблица 5.4 – Деструктуризация

|  |
| --- |
| Исходный код JavaScript |
| function renderRates(data) {  const { Date: date, Valute: valutes } = data; // Деструктуризация  dateSpan.textContent = new Date(date).toLocaleString('ru-RU');  ratesContainer.innerHTML = '';  for (const key in valutes) {  // Вложенная деструктуризация  const { CharCode, Name, Value, Nominal } = valutes[key];  const valuteElement = document.createElement('div');  valuteElement.className = 'valute-item';  // Используем шаблонную строку  valuteElement.innerHTML = `  <strong>${CharCode}</strong> (${Name}):  <span>${Value.toFixed(2)} руб. за ${Nominal} шт.</span>  `;  ratesContainer.appendChild(valuteElement);  }  } |

1.6. Добавьте обработчик события click на кнопку load-rates-btn, который будет вызывать вашу асинхронную функцию fetchAndRenderRates:

loadBtn.addEventListener('click', fetchAndRenderRates);

1.7. Проверьте результат в браузере. По нажатию на кнопку страница должна заполняться актуальными курсами валют.

2. Отображение графика динамики курса и обработка JSON

На этом шаге вы добавите возможность визуализировать изменение курса валюты за период и научитесь обрабатывать более сложные ответы JSON.

2.1. В файле index.html добавьте разметку для нового блока: выпадающие списки, кнопку и элемент <canvas>, на котором будет рисоваться график.

Таблица 5.5 – Обновленный HTML-код интерфейса

|  |
| --- |
| Исходный код HTML |
| <hr>  <h2>Динамика курса</h2>  <select id="currency-select">  <option value="USD">Доллар США</option>  <option value="EUR">Евро</option>  <option value="CNY">Китайский юань</option>  </select>  <select id="period-select">  <option value="7">Последняя неделя</option>  <option value="30">Последний ме-сяц</option>  </select>  <button id="show-chart-btn">Показать график</button>  <div><canvas id="currency-chart"></canvas></div> |

2.2. В main.js добавьте обработчик события для кнопки show-chart-btn, который будет вызывать новую асинхронную функцию fetchAndDrawChart.

2.3. Создайте функцию fetchAndDrawChart. В ней вы будете получать данные о курсах за период. Для получения исторических данных используем API frankfurter.app.

2.4. Внутри fetchAndDrawChart:

* Получите значения из выпадающих списков.
* Вычислите начальную и конечную даты.
* Сформируйте URL, используя шаблонные строки.
* Выполните fetch-запрос. Поскольку ответ сервера может содержать специальные типы данных, которые стандартный JSON.parse обработает некорректно (например, очень большие числа или даты в нестандартном формате), мы применим второй аргумент JSON.parse — **функцию-reviver** (восстановитель). Эта функция будет вызываться для каждой пары "ключ-значение" при разборе JSON и позволит нам вручную преобразовать данные.

2.5. Для работы с большими числами в JavaScript существует тип BigInt. Мы будем использовать его для демонстрации работы reviver-функции.

Таблица 5.6 – Функция отрисовки

|  |
| --- |
| Исходный код JavaScript |
| async function fetchAndDrawChart() {  const currency = document.getElementById('currency-select').value;  const days = document.getElementById('period-select').value;  const endDate = new Date();  const startDate = new Date();  startDate.setDate(endDate.getDate() - days);  const formattedStartDate = startDate.toISOString().split('T')[0];  const formattedEndDate = endDate.toISOString().split('T')[0];  const url = `https://api.frankfurter.app/${formattedStartDate}..  ${formattedEndDate}?from=${currency}&to=RUB`;  try {  const response = await fetch(url);  if (!response.ok) throw new Error('Ошибка сети');  const jsonText = await response.text();  // Используем функцию-reviver для обработки данных  const data = JSON.parse(jsonText, (key, value) => {  // Преобразуем строковые даты в объекты Date  if (key === 'start\_date' || key === 'end\_date' || !isNaN(Date.parse(key))) {  return new Date(key);  }  // Пример преобразования числовых значений в BigInt, если бы они были  if (typeof value === 'number' && value > Number.MAX\_SAFE\_INTEGER) {  return BigInt(value);  }  return value;  });  drawChart(data, currency);  } catch (error) {  console.error("Ошибка при получении данных для графика:", error);  }  } |

2.6. Создайте функцию drawChart(data, currency). Она будет обрабатывать данные и использовать библиотеку Chart.js для создания графика.

Таблица 5.7 – Функция экземпляра графика

|  |
| --- |
| Исходный код JavaScript |
| let myChart; // Переменная для хранения экземпляра графика  function drawChart(data, currency) {  const rates = data.rates;  const labels = Object.keys(rates).sort((a, b) => new Date(a) - new Date(b));  const values = labels.map(label => rates[label]['RUB']);  const ctx = document.getElementById('currency-chart').getContext('2d');  if (myChart) {  myChart.destroy();  }  myChart = new Chart(ctx, {  type: 'line',  data: {  labels: labels.map(d => new Date(d).toLocaleDateString('ru-RU')),  datasets: [{  label: `Курс ${currency} к RUB`,  data: values,  borderColor: 'rgba(75, 192, 192, 1)',  borderWidth: 2,  fill: false  }]  }  });  } |

2.7. Добавьте обработчик для кнопки show-chart-btn.

Таблица 5.8 – Функция экземпляра графика

|  |
| --- |
| Исходный код JavaScript |
| const showChartBtn = document.getElementById('show-chart-btn');  showChartBtn.addEventListener('click', fetchAndDrawChart); |

3. Получение погоды по геолокации пользователя

На этом шаге вы научитесь работать со встроенным в браузер Geolocation API для определения местоположения пользователя и последующего запроса данных о погоде для полученных координат.

3.1. В файле index.html после блока с графиком добавьте новую секцию.

Таблица 5.9 – Обновленный HTML-код интерфейса

|  |
| --- |
| Исходный код HTML |
| <hr>  <h2>Погода по вашему местоположению</h2>  <button id="get-weather-btn">Узнать погоду</button>  <div id="weather-container"></div> |

3.2. В main.js получите ссылки на новые элементы и добавьте обработчик события для кнопки.

3.3. Создайте функцию getWeatherByLocation. Внутри нее проверьте, поддерживает ли браузер геолокацию, и вызовите метод navigator.geolocation.getCurrentPosition(). Этот метод работает по принципу обратных вызовов (callbacks): он принимает две функции в качестве аргументов — для успеха и для ошибки.

Таблица 5.10 – Функция геолокации

|  |
| --- |
| Исходный код JavaScript |
| function getWeatherByLocation() {  if ("geolocation" in navigator) {  weatherContainer.textContent = 'Определение местоположения...';  navigator.geolocation.getCurrentPosition(handleLocationSuccess, handleLocationError);  } else {  weatherContainer.textContent = 'Браузер не поддерживает.';  }  } |

3.4. Создайте функцию handleLocationSuccess(position). Она будет вызвана в случае успеха и получит объект position. Используя деструктуризацию, извлеките из него широту и долготу и передайте их в функцию для запроса погоды.

Таблица 5.11 – Функция вызова погоды по координатам

|  |
| --- |
| Исходный код JavaScript |
| function handleLocationSuccess(position) {  const { latitude, longitude } = position.coords;  fetchWeather(latitude, longitude);  } |

3.5. Создайте функцию handleLocationError(error) для обработки возможных ошибок, например, если пользователь запретил доступ к своему местоположению.

3.6. Создайте асинхронную функцию fetchWeather(latitude, longitude). Она будет формировать URL для API погоды Open-Meteo, который не требует ключа аутентификации, и запрашивать данные.

Таблица 5.12 – Функция получения погоды

|  |
| --- |
| Исходный код JavaScript |
| async function fetchWeather(latitude, longitude) {  const url = `https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitude}&current\_weather=true`;  try {  const response = await fetch(url);  if (!response.ok) throw new Error('Ошибка при запросе погоды');  const data = await response.json();  renderWeather(data);  } catch (error) {  console.error('Ошибка:', error);  weatherContainer.textContent = 'Не удалось получить данные о погоде.';  }  } |

3.7. Создайте функцию renderWeather(data), которая отобразит полученную информацию на странице.

Таблица 5.13 – Функция отображения погоды

|  |
| --- |
| Исходный код JavaScript |
| function renderWeather(data) {  const { temperature, windspeed } = data.current\_weather;  weatherContainer.innerHTML = `  <h3>Текущая погода</h3>  <p>Температура: ${temperature} °C</p>  <p>Скорость ветра: ${windspeed} км/ч</p>  `;  } |

3.8. Проверьте работу. Браузер запросит у вас разрешение на доступ к геолокации. После согласия на странице должна появиться информация о погоде.

Заметки для одной из будущих лабораторных

При сохранении индекса в качестве атрибута преобразуйте этот индекс (очевидно, числовой) в строку с помощью конструкции шаблонной строки (аналог f-строк в Python): `${число}` (обратите внимание, что кавычки заменяются на символ обратной кавычки (выбирается на английской раскладке нажатием клавиши «Ё» на клавиатуре).