МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное
автономное образовательное
учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

«Введение во frontend разработку»

Направление подготовки 09.03.04 «Программная инженерия»
Направленность (профиль) «Разработка и сопровождение программного обеспечения»
Квалификация выпускника: бакалавр

Ставрополь

2026

Введение

Цель изучения дисциплины «Введение во frontend разработку» является формирование у студентов практических навыков создания клиентской части веб-приложений с использованием современных технологий (HTML, CSS, JavaScript), приобретение опыта верстки веб-интерфейсов, программирования интерактивного поведения страниц и основ взаимодействия с сервером, а также закрепление теоретических знаний, полученных в рамках дисциплины «Информационные технологии и программирование».

Задачами обучения являются:

- сформировать практические навыки семантической верстки и стилизации веб-страниц с использованием HTML5 и CSS3 (включая Flexbox, позиционирование, Box Model);
- освоить базовый синтаксис и конструкции языка JavaScript (переменные, типы данных, функции, условия, циклы, массивы, объекты);
- научить студентов работать с DOM (Document Object Model) и обрабатывать события для создания интерактивных интерфейсов;
- привить навыки использования инструментов разработчика (браузерные DevTools) и систем контроля версий (Git) для ведения проектов;
- познакомить с основами асинхронного программирования и выполнения HTTP-запросов к API (Fetch) для получения и отображения данных;
- сформировать понимание клиентского хранения данных (LocalStorage) и базовых принципов юзабилити (UI/UX) при проектировании интерфейсов;
- подготовить студентов к дальнейшему изучению фреймворков и более сложных дисциплин веб-разработки.

Лабораторная работа № 1

Инструментарий фронтенд-разработчика

Цель: Установка ПО (VS Code, браузер). Знакомство с DevTools. Структура HTML-документа. Создание простейшей страницы "Hello, World!".

Теоретическое обоснование

Современный инструментарий фронтендера
Фронтенд-разработка невозможна без правильно настроенной среды. Основные компоненты:

1. **Редактор кода** – программа для написания и редактирования исходного кода. VS Code (Visual Studio Code) – самый популярный редактор благодаря:
    - Подсветке синтаксиса HTML, CSS, JS.
    - Расширениям (Live Server, Prettier, ESLint, IntelliSense).
    - Встроенному терминалу и отладчику.
    - Поддержке Git.
2. **Браузер** – среда выполнения веб-страниц. Рекомендуются браузеры на движке Chromium (Chrome, Edge) или Firefox. Важнейшая часть – **DevTools** (инструменты разработчика), которые вызываются клавишей F12 или правым кликом → «Исследовать элемент». Основные вкладки:
    - **Elements** – просмотр и редактирование HTML/CSS в реальном времени.
    - **Console** – вывод сообщений JavaScript, выполнение команд.
    - **Network** – анализ сетевых запросов, времени загрузки.
    - **Sources** – отладка JavaScript (точки остановки).
    - **Application** – работа с localStorage, cookies.
3. **Структура HTML-документа**
   HTML (HyperText Markup Language) – язык разметки. Минимальный каркас:

```
html


<!DOCTYPE html>        <!-- Объявляет тип документа (HTML5) -->
<html lang="ru">       <!-- Корневой элемент -->
  <head>               <!-- Метаданные, не отображается -->
    <meta charset="UTF-8">   <!-- Кодировка символов -->
    <title>Заголовок вкладки</title>
  </head>
  <body>               <!-- Видимое содержимое -->
    <h1>Hello, World!</h1>
  </body>
</html>
```

- <!DOCTYPE html> – обязательно, иначе браузер переходит в режим совместимости (quirks mode).
- <head> содержит метатеги, заголовок, подключение стилей/скриптов.
- <body> – всё, что видит пользователь.

4. **Создание первой страницы** – задача «Hello, World!» проверяет работоспособность всех инструментов. Файл сохраняется с расширением .html. Открытие в браузере (двойным щелчком или через Live Server) позволяет сразу увидеть результат. Live Server автоматически обновляет страницу при изменениях.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Установите VS Code (официальный сайт) и браузер Google Chrome (или Firefox). Создайте папку lab1 на рабочем столе. В VS Code откройте эту папку, создайте файл index.html. Напишите базовую структуру HTML-документа, внутри <body> разместите фразу «Hello, World!». Сохраните файл. Откройте его в браузере двойным щелчком. Убедитесь, что надпись отображается.
2	Откройте созданную страницу в браузере, вызовите DevTools (F12 или Ctrl+Shift+I). Перейдите на

	вкладку **Elements**. Найдите узел <body> или <h1> (если использовали). Дважды кликните по тексту «Hello, World!» и измените его на «Мой первый HTML». Изменение должно сразу отобразиться на странице. Сделайте скриншот.
3	Дополните HTML-файл из задания 1: в секции <head> добавьте метатег <meta charset="UTF-8"> и тег <title> с текстом «Страница студента [Ваша фамилия и имя]». Проверьте, что вкладка браузера отображает правильный заголовок.
4	Создайте страницу, которая в <body> содержит три разных приветствия на русском, английском и немецком языках (например, «Привет!», «Hello!», «Hallo!»). Каждое приветствие должно быть оформлено в отдельном абзаце <p>. Добавьте горизонтальную линию <hr> между ними.
5	Используя DevTools (вкладка **Elements** → справа вкладка **Computed** или просто наведение на элемент), измерьте размеры (ширину и высоту) элемента <body> на вашей странице. Также определите размеры видимой области окна (viewport) – это можно увидеть вверху страницы при включённом режиме адаптивного дизайна (Toggle device toolbar). Запишите значения.
6	В VS Code установите следующие расширения: • Russian Language Pack (для русификации интерфейса), • Live Server (для автоматического обновления страницы), • Prettier (для форматирования кода). После установки перезапустите VS Code. Откройте ваш index.html, щёлкните правой кнопкой мыши по файлу и выберите «Open with Live Server». Убедитесь, что страница открылась в браузере и при изменении текста в HTML автоматически обновляется.
7	Создайте HTML-файл, в котором с помощью HTML-комментариев (<!-- ... -->) объясните назначение следующих секций: <!DOCTYPE html>, <html>, <head>, <body>. Например:

	<!-- DOCTYPE сообщает браузеру версию HTML -->. Сохраните файл как structure.html.
8	Создайте страницу, где в <head> задан базовый URL: <base href="https://example.com/">. В теле страницы разместите ссылку Фото. При клике по ссылке браузер должен перейти по адресу https://example.com/images/photo.jpg (это не обязательно рабочий URL, просто проверка работы base). Откройте DevTools, наведите на ссылку – внизу браузера должен отобразиться полный URL.
9	Откройте вашу страницу «Hello, World!» в браузере, откройте DevTools, перейдите на вкладку **Network**. Обновите страницу (F5). Найдите в списке файл index.html. Запишите время загрузки (поле Time). Также найдите вкладку **Performance** и нажмите запись (record) – после остановки записи посмотрите, сколько времени заняла отрисовка страницы.
10	Создайте две страницы: page1.html и page2.html. На первой странице разместите ссылку на вторую, на второй – ссылку обратно на первую. Проверьте переходы. Обе страницы должны иметь заголовки (тег <title>), соответствующие их содержанию.

Содержание отчета и его форма

1) Титульный лист с названием работы, ФИО, группой.
2) Скриншот VS Code с открытым файлом index.html.
3) Скриншот браузера с отображённой страницей.
4) Скриншот DevTools с изменённым содержимым.
5) Листинг кода каждой созданной страницы.
6) Ответы на контрольные вопросы.
7) Вывод о настройке окружения.

Вопросы для защиты работы

1. Для чего нужен VS Code и какие его основные возможности?

2. Как открыть DevTools в браузере и для чего используются вкладки Elements, Console, Network?
3. Что такое <!DOCTYPE html> и что будет, если его не указать?
4. Какие теги обязательно должны быть внутри <html>?
5. Чем отличается содержимое <head> от <body>?
6. Что делает метатег <meta charset="UTF-8">?
7. Как с помощью DevTools временно изменить текст на странице?
8. Что такое Live Server и зачем он нужен?
9. Какие расширения VS Code полезны для фронтендера?
10. Как проверить, что HTML-документ соответствует стандартам?

Критерии оценивания

Параметр	Баллы
Правильность установки ПО и настройки расширений	2
Корректная структура HTML-документов во всех заданиях	3
Наличие и качество скриншотов (DevTools, Live Server)	2
Полнота ответов на вопросы	2
Оформление отчёта	1

Лабораторная работа № 2

Основы разметки HTML

Цель: Теги, атрибуты, семантическая верстка. Понятие блочных и строчных элементов. Верстка простого текстового документа с заголовками, абзацами и списками.

Теоретическое обоснование

Теги и атрибуты

HTML-документ состоит из элементов, которые обозначаются тегами.

Большинство тегов парные: текст . Есть и одиночные:
, <img>.

Атрибуты уточняют элемент: Ссылка.

- href – адрес перехода.
- target="_blank" – открыть в новой вкладке.
- class – для группировки элементов (CSS/JS).
- id – уникальный идентификатор.

Семантическая вёрстка – использование тегов, которые несут смысл, а не просто «контейнеры»:

- <header> – шапка сайта.
- <nav> – навигационное меню.
- <main> – основное содержимое (уникальное для страницы).
- <article> – самостоятельный блок (пост, новость).
- <section> – раздел тематической группы.
- <aside> – боковая панель, дополнительная информация.
- <footer> – подвал.

Семантика улучшает доступность (screen readers) и SEO.

Блочные и строчные элементы

- **Блочные** (, , <h1>–<h6>, <ul>, <li>) – занимают всю доступную ширину, начинаются с новой строки. Можно задавать ширину, высоту, отступы.
- **Строчные** (, <a>, , , <img>) – располагаются в строке, не переносятся, ширина по содержимому. Нельзя задать width/height (но можно margin-left/right).

Вёрстка текстового документа

Заголовки (<h1>–<h6>) – иерархия: один <h1> на странице.

Абзацы – .

Списки:

- Маркированный <ul> – элемент.
- Нумерованный <ol> – .

- Список определений <dl> – <dt> (термин) и <dd> (описание).

Пример простой страницы:

```
html


<article>
  <h1>Рецепт блинов</h1>
  <p>Классический рецепт...</p>
  <h2>Ингредиенты</h2>
  <ul><li>Молоко</li><li>Яйца</li></ul>
  <h2>Приготовление</h2>
  <ol><li>Смешать</li><li>Жарить</li></ol>
</article>
```

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте страницу, на которой присутствуют все заголовки от <h1> до <h6>. Под каждым заголовком напишите пояснение, какой уровень заголовка и где он обычно используется (например, <h1> – главный заголовок страницы).
2	Напишите абзац текста (5–6 предложений на любую тему). Внутри абзаца выделите жирным шрифтом () два ключевых слова, а курсивом () – одно важное понятие. Также добавьте одно слово, подчёркнутое с помощью <ins> (вставка) и одно перечёркнутое <del> (удаление).
3	Создайте нумерованный список «План обучения на семестр» из 5 пунктов (например, «Изучить HTML», «Освоить CSS» и т.д.). Ниже создайте маркированный список «Используемые инструменты» (VS Code, Git, Chrome DevTools и др.). Вложите в один из пунктов маркированного списка дополнительный вложенный список (например, подпункты для VS Code: расширения).
4	Используя семантические теги, сверстайте структуру простого сайта:

	• <header> – содержит название сайта «Мой блог» и навигацию (<nav> со ссылками «Главная», «Обо мне», «Контакты»). • <main> – содержит статью (<article>) с заголовком и двумя абзацами. • <footer> – содержит копирайт и год.
5	Вставьте на страницу длинную цитату (3–4 строки) из известного произведения, используя тег <blockquote>. Под цитатой укажите автора с помощью <cite>. Также добавьте короткую строчную цитату внутри абзаца с помощью <q>.
6	Создайте список определений (<dl>) из 3 терминов по фронтенд-разработке. Например: <dt>HTML</dt><dd>Язык разметки гипертекста</dd> и т.д. Оформите красиво.
7	Добавьте на страницу несколько строчных элементов . Каждому задайте свой цвет с помощью атрибута style (например, красный). Используйте разные цвета: красный, зелёный, синий. Объясните, чем строчные элементы отличаются от блочных.
8	Сверстайте рецепт любимого блюда. Обязательные элементы: • Заголовок блюда (<h1>). • Маркированный список ингредиентов. • Нумерованный список шагов приготовления. • Абзац с примечанием (например, «Совет: подавать горячим»). • Изображение блюда (можно использовать картинку-заглушку из интернета).
9	Создайте страницу, на которой внутри блочного элемента размещены три строчные ссылки (<a>) на ваши любимые сайты (например, Google, YouTube, GitHub). Ссылки должны открываться в новой вкладке (target="_blank"). Между ссылками поставьте вертикальную черту | или другой разделитель.
10	Используя тег <pre>, отформатируйте стихотворение (4–6 строк) с сохранением оригинальных пробелов и переносов строк. Также внутри <pre> можно использовать тег <code> для фрагмента кода (например,

	небольшой CSS-правило). Объясните, для чего нужен тег <pre>.

Содержание отчета и его форма

1) Титульный лист.
2) Скриншоты каждой выполненной страницы (или одной страницы со всеми заданиями).
3) Исходный HTML-код с пояснениями, где использованы блочные/строчные элементы.
4) Письменные ответы на вопросы.
5) Вывод о роли семантики.

Вопросы для защиты работы

1. Чем отличается блочный элемент от строчного? Приведите примеры.
2. Какие семантические теги вы знаете и для чего они нужны?
3. Что такое атрибут class и id? В чём разница?
4. Как создать вложенный список (список внутри списка)?
5. Для чего используется тег <article>?
6. Что делает тег
? К какому типу элементов он относится?
7. Как правильно оформить гиперссылку, открывающуюся в новой вкладке?
8. Чем отличается от <b>?
9. Как вставить специальные символы (например, ©)?
10. Что такое «семантическая верстка» и почему она важна?

Критерии оценивания

Параметр	Баллы
Корректное использование блочных и строчных тегов	2
Применение семантических тегов	2

Разнообразие списков и вложенных структур	2
Ответы на контрольные вопросы	2
Оформление отчёта	2

Лабораторная работа № 3
Основы стилизации CSS

Цель и содержание работы: Селекторы, свойства, базовые стили (цвета, шрифты, текст). Подключение стилей к HTML.

Теоретическое обоснование

CSS (Cascading Style Sheets) – язык описания внешнего вида. **Способы подключения**:

1. **Внешний** (рекомендуется) – отдельный файл style.css, подключается через <link rel="stylesheet" href="style.css"> в <head>.
2. **Внутренний** – блок <style> внутри <head>.
3. **Inline** – атрибут style="color: red;" у элемента (имеет наивысший приоритет, но плохо поддерживается).

Селекторы – правила, по которым CSS выбирает элементы:

- Селектор тега: p { }
- Селектор класса: .my-class { }
- Селектор id: #unique { } (использовать редко)
- Универсальный селектор: * { }
- Группировка: h1, h2, h3 { }
- Вложенный: article p { } (все p внутри article)
- Псевдоклассы: a:hover { }, li:first-child { }

Базовые свойства:

- Цвет текста: color: #ff0000; (можно имена, HEX, RGB, HSL).
- Фон: background-color: lightgray;.
- Шрифт: font-family: Arial, sans-serif; (несколько названий – fallback).

- Размер шрифта: font-size: 16px; (px, em, rem, %).
- Начертание: font-weight: bold;, font-style: italic;.
- Выравнивание: text-align: center; left; right; justify.
- Высота строки: line-height: 1.5;.

Приоритет CSS (от меньшего к большему):

1. Стили браузера по умолчанию.
2. Внешний/внутренний CSS (порядок подключения важен).
3. Селекторы: тег < класс < id < inline-стили.
4. !important – ломает каскад, использовать с осторожностью.

DevTools для CSS – вкладка Elements → Styles показывает применённые правила, можно включать/отключать свойства, добавлять новые, видеть размеры элементов.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте внешний CSS-файл style.css. Подключите его к любой странице из предыдущих работ (например, к странице с рецептом). В CSS задайте для тега body цвет фона #f0f0f0 и шрифт Arial, sans-serif.
2	Для всех заголовков h1, h2, h3 установите: • Шрифт Georgia, • Цвет darkblue, • Выравнивание по центру. Для h1 дополнительно задайте размер 32px, для h2 – 28px.
3	Для всех абзацев (p) задайте: • Шрифт Verdana, • Размер 16px, • Цвет #333, • Нижний отступ (margin-bottom) 15px. Для первого абзаца после каждого заголовка добавьте отступ сверху 10px (используйте соседний селектор h1 + p, h2 + p и т.д.).

4	Создайте класс .highlight. Примените его к нескольким фрагментам текста в разных местах страницы. Стиль класса: жёлтый фон (background-color: yellow), полужирный шрифт, курсив.
5	Найдите или создайте абзац с id="special". Оформите его с помощью селектора #special: • Рамка 2px solid red, • Внутренний отступ 10px, • Фон lightyellow.
6	Для всех ссылок (<a>) уберите подчёркивание (text-decoration: none), задайте цвет green. При наведении мыши (:hover) измените цвет на orange и добавьте подчёркивание. Для посещённых ссылок (:visited) задайте цвет gray.
7	Для маркированного списка ( ) измените стиль маркера на квадрат (list-style-type: square). Также задайте цвет маркеров (используйте ::marker или измените цвет текста списка – маркеры унаследуют цвет). Для нумерованного списка задайте римские цифры (list-style-type: upper-roman).
8	Используя псевдокласс :first-of-type, выберите первый абзац внутри <article> и задайте ему отступ слева 30px и курсивное начертание. Сделайте так, чтобы этот стиль не влиял на другие абзацы.
9	Создайте класс .card со следующими свойствами: • Белый фон, • Тень (box-shadow: 0 4px 8px rgba(0,0,0,0.1)), • Скругление углов 8px, • Внутренний отступ 20px, • Ширина 300px. Примените этот класс к блоку с рецептом или карточке товара из предыдущих работ. Убедитесь, что карточка выглядит как отдельный элемент.
10	Эксперимент с приоритетом: • Создайте страницу, подключите внешний CSS, где для параграфов задан красный цвет. • Внутри <head> добавьте внутренний стиль <style> для тех же параграфов – зелёный цвет. • В одном из параграфов используйте inline-

	стиль style="color: blue". • Посмотрите, какой цвет победит. Затем закомментируйте внутренний стиль и проверьте. Сделайте вывод.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) Код HTML (из ЛР2) и CSS.
3) Скриншот оформленной страницы.
4) Скриншот DevTools с вкладкой Styles, показывающий применённые правила.
5) Ответы на вопросы.
6) Вывод о важности внешних таблиц стилей.

Вопросы для защиты работы

1. Какие существуют способы подключения CSS к HTML? Их преимущества и недостатки.
2. Что такое селектор? Назовите основные виды.
3. Какой селектор имеет наивысший приоритет: `#id`, `.class` или `тег`?
4. Как задать цвет текста и фона элемента?
5. Какие единицы измерения шрифта вы знаете (px, em, rem)?
6. Что такое псевдокласс `:hover`? Приведите пример.
7. Как сделать текст полужирным и курсивным через CSS?
8. Как убрать маркеры у списка?
9. Что такое каскадность CSS и как она работает?
10. Как отладить CSS через DevTools (изменять правила, отключать свойства)?

Критерии оценивания

Параметр	Баллы
Правильное подключение CSS (внешний файл)	2
Использование различных селекторов (тег, класс, id, псевдокласс)	3
Эстетичность и читаемость стилей	2
Ответы на контрольные вопросы	2
Оформление отчета	1

Лабораторная работа 4

Модель расположения элементов (CSS Box Model)

Цель: Работа с отступами (margin, padding), рамками (border), размерами (width, height). Создание карточки товара или визитной карточки.

Теоретическое обоснование

Box Model – каждый элемент на странице представляет собой прямоугольник, состоящий из четырёх слоёв (от внутреннего к внешнему):

1. **Content** – содержимое (текст, картинка). Ширина и высота задаются width / height.
2. **Padding** – внутренний отступ от контента до границы. Свойства: padding-top, padding-right, padding-bottom, padding-left или сокращённо padding: 10px 20px; (верх/низ, право/лево).
3. **Border** – рамка. border-width, border-style (solid, dotted, dashed), border-color. Сокращённо border: 1px solid black;.
4. **Margin** – внешний отступ от рамки до соседних элементов. margin: 10px; – отступ со всех сторон.

Схлопывание вертикальных отступов – если у двух соседних блочных элементов заданы margin-bottom и margin-top, то они схлопываются в один,

равный большему из значений. Горизонтальные отступы не схлопываются.

Свойство box-sizing:

- box-sizing: content-box (по умолчанию) – ширина задаётся только для content; padding и border увеличивают общую ширину элемента.
- box-sizing: border-box – ширина включает content, padding и border. Удобно для создания сеток, т.к. легче рассчитывать размеры.

Пример:

```css
.card {
  width: 300px;
  padding: 20px;
  border: 1px solid #ccc;
  margin: 15px;
  box-sizing: border-box; /* реальная ширина останется 300px */
}
```

Закругление углов – border-radius: 10px; (можно задать отдельные углы). **Тень** – box-shadow: 5px 5px 10px rgba(0,0,0,0.2); (смещение по X, по Y, размытие, цвет).

Практика – создание карточки товара требует понимания всех слоёв box model, чтобы правильно разместить изображение, цену, кнопку.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте с классом .box. Задайте ему: • ширину 300px, • высоту 200px, • рамку 2px solid red, • внутренний отступ 20px со всех сторон. Вычислите и запишите реальную ширину элемента (ширина контента + левый/правый padding + левая/правая рамка). Проверьте с помощью DevTools.
2	Создайте карточку товара с классом .card. Задайте: • width: 250px, • padding: 15px,

	• border: 1px solid #ccc, • box-sizing: border-box. Убедитесь, что общая ширина карточки осталась 250px. Для проверки задайте фоновый цвет и посмотрите на реальные размеры.
3	Создайте родительский контейнер (ширина 800px, фон светло-серый). Внутри разместите три карточки (из задания 2) в ряд. Для карточек задайте display: inline-block и margin-right: 20px. Выровняйте контейнер по центру страницы с помощью margin: 0 auto. Сделайте так, чтобы у последней карточки не было правого отступа.
4	Добавьте карточке тень (box-shadow: 5px 5px 15px rgba(0,0,0,0.3)) и скругление углов (border-radius: 15px). При наведении на карточку увеличивайте тень (например, box-shadow: 10px 10px 20px rgba(0,0,0,0.4)). Сделайте плавный переход (transition: box-shadow 0.3s).
5	Создайте визитную карточку (дизайн на ваш выбор). Она должна содержать: • Фото (заглушка или иконка), • Имя и фамилию, • Должность, • Контакты (email, телефон). Используйте margin для отступов между элементами, padding – для внутренних отступов карточки. Ширина карточки – 300px.
6	Эксперимент со схлопыванием отступов: • Создайте два блока один под другим. • Первому задайте margin-bottom: 30px, второму – margin-top: 20px. • В DevTools посмотрите расстояние между блоками. Оно должно быть 30px (большее значение). • Затем добавьте между ними пустой блок с overflow: auto или border: 1px solid transparent, чтобы предотвратить схлопывание. Сравните.
7	Создайте карточку товара с изображением (любая картинка из интернета), названием товара, ценой и кнопкой «Купить». При наведении на карточку увеличивайте её тень и добавляйте небольшую

	анимацию (например, масштабирование transform: scale(1.02)).
8	Используя отрицательный margin, наложите один блок на другой. Например, создайте два квадрата 100×100 с разными фонами. Второму задайте margin-top: -50px. Поэкспериментируйте с z-index, чтобы управлять порядком перекрытия.
9	Для родительского контейнера (в котором находятся три карточки) примените display: flex (без глубокого изучения) со свойствами justify-content: space-between и align-items: center. Разместите карточки равномерно.
10	Создайте горизонтальную линейку из трёх цветных квадратов (красный, зелёный, синий). Каждый квадрат: width: 100px, height: 100px, margin: 5px. Сделайте так, чтобы они располагались в одну строку с помощью display: inline-block. Уберите лишние пробелы между квадратами (например, задав родительскому контейнеру font-size: 0, а квадратам вернуть нужный font-size).

Содержание отчета и его форма

1) Титульник.
2) Код HTML/CSS карточки товара/визитки.
3) Скриншот результата.
4) Скриншот DevTools с визуализацией box model.
5) Ответы на вопросы.
6) Вывод о применении box model.

Вопросы для защиты работы

1. Из каких слоёв состоит box-модель?
2. В чём разница между margin и padding?
3. Как box-sizing: border-box влияет на вычисление ширины?
4. Почему происходит схлопывание вертикальных margin?

5. Как сделать элемент круглым с помощью border-radius?
6. Какие значения может принимать свойство display (block, inline, inline-block)?
7. Как задать рамку только снизу (border-bottom)?
8. Что такое outline и чем отличается от border?
9. Как центрировать блочный элемент горизонтально?
10. Как измерить реальные размеры элемента в DevTools?

Критерии оценивания

Параметр	Баллы
Корректное использование margin, padding, border	2
Понимание box-sizing	2
Вёрстка карточки (эстетика, адаптивность в пределах задания)	3
Ответы на вопросы	2
Оформление отчета	1

Лабораторная работа № 5

Git для фронтендера

Цель работы: Основы работы в терминале. Инициализация репозитория, коммиты, ветки. Публикация созданной страницы на GitHub Pages.

Теоретическое обоснование

Git – распределённая система контроля версий. Позволяет отслеживать изменения, возвращаться к предыдущим версиям, работать в команде.

Основные понятия:

- **Репозиторий** – хранилище проекта (папка с файлами и скрытая папка .git).
- **Коммит** – снимок состояния файлов в определённый момент. Каждый коммит имеет уникальный хеш и сообщение.

- **Ветка (branch)** – независимая линия разработки. Основная ветка часто называется main или master.
- **Удалённый репозиторий** – хранилище на сервере (GitHub, GitLab, Bitbucket).

Базовые команды (работа в терминале):

- git init – инициализировать репозиторий в текущей папке.
- git status – посмотреть состояние (изменённые, добавленные файлы).
- git add index.html – добавить файл в staging area (подготовить к коммиту). git add . – добавить все изменённые файлы.
- git commit -m "сообщение" – создать коммит.
- git log – история коммитов.
- git branch feature – создать ветку feature.
- git checkout feature – переключиться на ветку feature (или git switch feature).
- git merge feature – влить ветку feature в текущую.
- git remote add origin https://github.com/username/repo.git – связать локальный репозиторий с удалённым.
- git push -u origin main – отправить коммиты на удалённый репозиторий.
- git pull – получить изменения с удалённого репозитория.

GitHub Pages – сервис для хостинга статических сайтов. В настройках репозитория нужно выбрать ветку и папку (обычно /root), после чего сайт становится доступен по адресу https://username.github.io/repo-name/.

Практическая ценность – публикация лабораторных работ в интернете, возможность демонстрации портфолио, совместная разработка.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	На рабочем столе создайте папку git-lab. Откройте терминал (в VS Code: Ctrl+). Перейдите в эту папку (cd git-lab). Выполните командуgit init. Убедитесь, что появилась скрытая папка.git`.

2	Создайте в папке файл index.html с содержимым любой страницы из предыдущих ЛР (например, визитка). Выполните git add index.html, затем git commit -m "first commit". Посмотрите статус git status.
3	Создайте ветку develop командой git branch develop. Переключитесь на неё: git checkout develop. Измените заголовок страницы на «Разработка» и добавьте новый абзац. Сделайте коммит с сообщением «update title and content».
4	Вернитесь в ветку main (git checkout main). Выполните слияние ветки develop в main: git merge develop. Убедитесь, что изменения появились в файле. Если возник конфликт (маловероятно), разрешите его.
5	Создайте удалённый репозиторий на GitHub (зайдите на github.com, нажмите «New repository», назовите my-first-repo, не добавляйте README). Свяжите локальный репозиторий с удалённым: git remote add origin https://github.com/ваш_логин/my-first-repo.git (замените на ваш URL).
6	Отправьте изменения в удалённый репозиторий: git push -u origin main. Обновите страницу GitHub – файл index.html должен появиться.
7	В настройках репозитория GitHub перейдите в раздел **Pages**. В качестве источника выберите ветку main и папку / (root). Нажмите Save. Через минуту сайт будет доступен по ссылке https://ваш_логин.github.io/my-first-repo/. Откройте эту ссылку в браузере. Убедитесь, что страница работает.
8	Создайте новую ветку feature от main: git checkout -b feature. Добавьте на страницу новый раздел (например, «Мои навыки» со списком). Сделайте коммит и запушите ветку на GitHub: git push -u origin feature. Зайдите на GitHub, перейдите на вкладку **Pull requests** и создайте Pull Request из feature в main. В комментарии напишите «Добавлен раздел навыков». Слейте PR через веб-интерфейс.
9	В файле index.html намеренно сделайте опечатку (например, в слове «Привет»). Сделайте коммит с

	сообщением «fix: исправлена опечатка». Выполните git log --oneline, чтобы увидеть историю коммитов. Найдите хеш предыдущего коммита и вернитесь к нему с помощью git checkout <hash> (просто для просмотра, затем вернитесь в main).
10	Склонируйте свой репозиторий в другую папку на компьютере (или на другой компьютер): git clone https://github.com/ваш_логин/my-first-repo.git. Внесите изменения (например, добавьте ещё один абзац), сделайте коммит и запушите. Затем в исходной папке выполните git pull, чтобы получить изменения.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, вариант.
2) Скриншоты терминала с вводом команд (git init, add, commit, branch, push).
3) Ссылка на репозиторий на GitHub.
4) Ссылка на работающий сайт на GitHub Pages.
5) Скриншот страницы GitHub Pages.
6) Ответы на вопросы.
7) Вывод о роли Git в разработке.

Вопросы для защиты работы

1. Что такое Git и для чего он нужен фронтендеру?
2. Чем отличается `git add .` от `git add index.html`?
3. Что такое коммит? Как написать хорошее сообщение коммита?
4. Для чего нужны ветки в Git?
5. Как переключиться на другую ветку?
6. Что делает команда `git merge`?
7. Как связать локальный репозиторий с удалённым?
8. Как опубликовать сайт на GitHub Pages?
9. Что такое Pull Request?

10.Как посмотреть историю коммитов и различия между версиями?

Критерии оценивания

Параметр	Баллы
Успешная инициализация и базовые операции (add, commit)	2
Работа с ветками и слияние	2
Публикация на GitHub Pages	3
Ответы на вопросы	2
Оформление отчета	1

Лабораторная работа № 6

Позиционирование элементов в CSS

Цель: Свойство position. Свойство display: flexbox. Базовая адаптивность. Верстка простого макета: шапка, контент, подвал.

Теоретическое обоснование

Свойство position определяет метод позиционирования элемента:

Значение	Относительно чего?	Поведение
static	(по умолчанию)	Элемент находится в обычном потоке. Свойства top/left не работают.
relative	своего нормального положения	Смещается с помощью top, left, но занимает исходное место в потоке.
absolute	ближайшего позиционированного родителя (не static)	Вырывается из потока, другие элементы его игнорируют.
fixed	окна браузера	Всегда на одном месте при прокрутке.
sticky	пересечение границы	Как relative, но при прокрутке прилипает к указанной позиции (top:0).

Примеры:

- Кнопка «Наверх» – position: fixed; bottom: 20px; right: 20px;.
- Выпадающее меню – родитель position: relative, дочернее меню position: absolute; top: 100%; left: 0;.

Flexbox – одномерная модель раскладки (в строку или в столбец). Свойства контейнера display: flex;:

- flex-direction: row (по умолчанию), column, row-reverse.
- justify-content – выравнивание по главной оси: flex-start, center, space-between, space-around.
- align-items – выравнивание по поперечной оси: stretch (по умолчанию), center, flex-start, flex-end.
- flex-wrap – перенос элементов на новую строку (wrap, nowrap).
- gap – расстояние между элементами.

Для элементов внутри flex-контейнера:

- flex-grow – коэффициент растяжения (1 – займёт всё доступное пространство).
- flex-shrink – сжатие.
- flex-basis – базовая ширина.

Адаптивность – медиа-запросы @media позволяют применять разные стили в зависимости от ширины экрана, ориентации и т.д.:

```css
@media (max-width: 768px) {
  .container { flex-direction: column; }
}
```

Типовой макет (шапка – контент – подвал) часто реализуется через body как flex-контейнер с flex-direction: column и min-height: 100vh. Основной контент получает flex: 1, чтобы прижимать подвал к низу.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте страницу с шапкой (<header>, высота 100px, фон серый), основным контентом (<main>, минимальная высота 400px) и подвалом (<footer>, высота 60px, фон тёмно-серый). Используйте для body display: flex, flex-direction: column, min-height: 100vh. Для main задайте flex: 1, чтобы подвал прижимался к низу.

2	В шапке разместите логотип (текст «Мой сайт») слева и навигационное меню (ссылки «Главная», «О нас», «Контакты») справа. Используйте display: flex для шапки, justify-content: space-between и align-items: center.
3	В основном контенте создайте контейнер для карточек товаров. Задайте ему display: flex, flex-wrap: wrap, gap: 20px, justify-content: center. Добавьте 4–6 карточек товаров (из ЛР4). Карточки должны иметь одинаковую ширину (250px) и переноситься на новую строку при нехватке места.
4	Внутри main создайте две колонки: левую боковую панель (ширина 200px) и основную область (занимает оставшееся пространство). Используйте display: flex. Боковой панели задайте фон #f4f4f4, основной области – #fff. Разместите в боковой панели несколько ссылок (категории).
5	Добавьте на страницу кнопку «Наверх» (<button> или <a>). Стилизуйте её: фиксированное положение (position: fixed), нижний правый угол (bottom: 20px, right: 20px), круглый фон, белый цвет. При клике по кнопке с помощью JS (можно встроить) прокрутите страницу вверх: window.scrollTo({top:0, behavior:'smooth'}).
6	Создайте выпадающее меню на основе position: absolute. Навигационный пункт «Услуги» при наведении показывает подменю (список услуг). Родительский пункт меню должен иметь position: relative, подменю – position: absolute, top: 100%, left: 0. Подменю по умолчанию скрыто (display: none), при наведении на родителя появляется (display: block).
7	Добавьте медиа-запрос: при ширине экрана менее 768px: • Шапка становится вертикальной (логотип и меню в колонку). • Боковая панель переносится под основное содержимое (используйте flex-wrap: wrap и измените порядок с помощью order). • Карточки занимают 100% ширины.
8	Создайте баннер (контейнер с фоновым изображением, высота 300px). Поверх изображения разместите текст

	заголовка и кнопку. Используйте position: relative для контейнера баннера, а для текста – position: absolute с центрированием (top: 50%, left: 50%, transform: translate(-50%, -50%)).
9	Реализуйте sticky-шапку: при прокрутке страницы шапка должна оставаться приклеенной к верху. Используйте position: sticky, top: 0. Убедитесь, что шапка не перекрывает контент (добавьте z-index).
10	Сверстайте упрощённый макет интернет-магазина: • Шапка (логотип, строка поиска, иконка корзины). • Каталог – сетка из карточек товаров (минимум 6). • Подвал (контакты, ссылки на соцсети). Всё должно быть адаптивно. Используйте только HTML и CSS (без JS). Примените Flexbox или Grid.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) Сверстайте упрощённый макет интернет-магазина:
3) Шапка (логотип, строка поиска, иконка корзины).
4) Каталог – сетка из карточек товаров (минимум 6).
5) Подвал (контакты, ссылки на соцсети).
6) Всё должно быть адаптивно. Используйте только HTML и CSS (без JS). Примените Flexbox или Grid.

Вопросы для защиты работы

1. Какие значения свойства position существуют? Чем отличается relative от absolute?
2. Как работает z-index?
3. Что такое Flexbox и для чего он нужен?
4. Как выровнять элементы по главной оси (justify-content) и поперечной (align-items)?
5. Что делает свойство flex-wrap?
6. Как создать две колонки одинаковой высоты с помощью Flexbox?

7. Что такое медиа-запрос (@media)? Приведите пример.
8. Как сделать элемент «липким» (sticky)?
9. В чём отличие position: fixed от sticky?
10. Как центрировать элемент по вертикали и горизонтали с помощью Flexbox?

Критерии оценивания

Параметр	Баллы
Корректная структура (шапка, контент, подвал)	2
Использование Flexbox для сетки и выравнивания	3
Применение position (absolute/fixed/sticky)	2
Адаптивность через медиа-запрос	2
Оформление отчета	1

Лабораторная работа № 7

Введение в JavaScript

Цель: Подключение скрипта к странице. Переменные (var, let, const), типы данных. Базовые операции и взаимодействие с пользователем (alert, prompt, console.log).

Теоретическое обоснование

JavaScript – высокоуровневый, интерпретируемый язык программирования, который работает в браузере и обеспечивает интерактивность.

Подключение к HTML:

- Внутри <head> или перед закрывающим </body>: <script src="script.js"></script>.
- Встроенный скрипт: <script> // код </script>.

Переменные:

- var – устаревший, имеет функциональную область видимости.
- let – современный, блочная область видимости (внутри {}).
- const – константа, не может быть переприсвоена, но объекты/массивы можно изменять.

Типы данных:

- number – целые и дробные числа.
- string – строки в кавычках (одинарных, двойных, обратных).
- boolean – true / false.
- null – «ничего», намеренное отсутствие значения.
- undefined – значение не присвоено.
- object – коллекции (массивы, объекты, даты).
- symbol (редко).

Взаимодействие с пользователем:

- alert("Сообщение") – модальное окно с сообщением.
- prompt("Вопрос", "значение по умолчанию") – возвращает введённую строку (или null при отмене).

- console.log("Вывод в консоль") – для отладки.

Базовые операции:

- Арифметические: +, -, *, /, % (остаток), ** (степень).
- Присваивание: =, +=, -= и т.д.
- Сравнение: == (с приведением типов), === (строгое, без приведения), !=, !==, <, >, <=, >=.
- Логические: && (И), || (ИЛИ), ! (НЕ).

Преобразование типов:

- В число: Number("123") или +"123".
- В строку: String(123) или 123 + "".
- В булево: Boolean(0) – false, Boolean("привет") – true.

Пример скрипта приветствия:

```
js

let name = prompt("Как вас зовут?");
alert("Привет, " + name + "!");
```

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте HTML-страницу. Внутри <body> добавьте пустой абзац с id="output". Подключите внешний script.js. В скрипте выведите в консоль строку «Скрипт работает». Также измените текст абзаца output на «JavaScript подключён».
2	Объявите переменную userName с помощью let. Используйте prompt для запроса имени пользователя. Затем выведите alert с сообщением «Привет, [имя]!». Если пользователь ничего не ввёл, используйте «Гость».
3	Объявите две числовые переменные a и b, присвойте им значения 15 и 7. Вычислите сумму, разность, произведение и частное (деление). Результаты выведите последовательно через alert или в консоль. Обратите внимание, что частное может быть дробным.
4	Объявите константу PI = 3.14159. Запросите у пользователя радиус круга (через prompt). Вычислите

	площадь круга по формуле PI * r * r. Выведите результат с точностью до двух знаков (используйте toFixed(2)).
5	Создайте переменные разных типов: let num = 42;, let str = "Hello";, let bool = true;, let nothing = null;, let undef;. Используя typeof, выведите в консоль тип каждой переменной. Объясните, почему typeof null возвращает object.
6	Напишите скрипт, который запрашивает у пользователя год рождения (число). Вычислите возраст (текущий год – год рождения). Если возраст меньше 0 или больше 120, выведите сообщение «Некорректный год». Иначе выведите «Ваш возраст: X лет».
7	Создайте страницу с кнопкой <button onclick="showMessage()">Нажми</button>. Внутри тега <script> (или во внешнем файле) объявите функцию showMessage(), которая вызывает alert("Hello!"). Убедитесь, что при клике на кнопку появляется окно.
8	Запросите у пользователя число через prompt. Преобразуйте его в число с помощью Number(). Проверьте, является ли введённое значение числом (функция isNaN). Если не число – выведите «Ошибка: введите число». Если число – выведите его квадрат.
9	Объявите три переменные: firstName = "Иван", lastName = "Петров", age = 30. Склейте их в одну строку вида «Иван Петров, 30 лет». Выведите результат с помощью alert или в консоль.
10	Создайте простой калькулятор: последовательно запрашивайте у пользователя первое число, оператор (+, -, *, /) и второе число. Выполните соответствующую операцию и выведите результат. Предусмотрите деление на ноль. Если оператор не распознан – сообщите об ошибке.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) HTML-файл с подключённым JS.
3) Скриншот работы скрипта в браузере (alert/prompt).

4) Скриншот консоли с console.log.
5) Листинг кода.
6) Ответы на вопросы.
7) Вывод о взаимодействии с пользователем.

Вопросы для защиты работы

1. Как подключить JavaScript к HTML?
2. Чем отличается let от const?
3. Какие примитивные типы данных существуют в JS?
4. Что выведет console.log(typeof null)?
5. Как преобразовать строку в число?
6. Для чего используется alert() и prompt()?
7. Что такое NaN и когда он возникает?
8. Как проверить, что переменная является числом?
9. Как вывести сообщение в консоль браузера?
10. В чём разница между == и ===?

Критерии оценивания

Параметр	Баллы
Правильное подключение скрипта	1
Использование переменных, констант, типов	2
Взаимодействие через alert/prompt	2
Выполнение вычислений	2
Ответы на вопросы	2
Оформление отчета	1

Лабораторная работа № 8

Управляющие конструкции в JS

Цель: Условные операторы (if/else, switch) и логические операторы.

Теоретическое обоснование

Условный оператор if:

```
js

if (условие) {
  // код, если true
} else if (другое условие) {
  // код
} else {
  // код, если всё false
}
```

Тернарный оператор – сокращённая форма: let status = age >= 18 ? "взрослый" : "ребёнок";

switch – для множественного сравнения:

```
js

switch(значение) {
  case 'apple':
    console.log('Яблоко');
    break;
  case 'banana':
    console.log('Банан');
    break;
  default:
    console.log('Неизвестно');
}
```

break предотвращает «проваливание» в следующий case.

Генерация случайного числа:

```
js

// Целое от min до max включительно
let random = Math.floor(Math.random() * (max - min + 1)) + min;
```

Игра «Угадай число»:

1. Генерируется число.
2. Пользователь вводит число.
3. Сравниваются: если больше – подсказка «меньше», если меньше – «больше».
4. Повторять попытки, пока не угадает или не кончатся попытки.

Логические операторы используются в условиях:

if (guess > secret && tries < maxTries)

Проверка на число – функция isNaN(value) возвращает true, если значение не число. После prompt результат всегда строка, поэтому нужно преобразование: let num = Number(input); if (isNaN(num)) { ... }.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Напишите скрипт, который запрашивает число. Если число больше 0 – выводит «положительное», если меньше 0 – «отрицательное», если равно 0 – «ноль». Используйте if/else.
2	Запросите возраст. Выведите категорию: 0–12 – «ребёнок», 13–17 – «подросток», 18–64 – «взрослый», 65+ – «пенсионер». Используйте if else if.
3	Используя switch, по номеру месяца (1–12) выведите название сезона (зима, весна, лето, осень). Если введено число вне диапазона – сообщение об ошибке.
4	Игра «Угадай число» (3 попытки): • Компьютер загадывает случайное число от 1 до 10. • Пользователь вводит число. • Если угадал – поздравление и выход из игры. • Если нет – подсказка «больше» или «меньше», количество попыток уменьшается. • Если попытки закончились – показать загаданное число. Используйте цикл for или while для ограничения попыток.
5	Добавьте в игру проверку на ввод не числа: если пользователь ввёл не число, вывести предупреждение и не засчитывать попытку (т.е. попросить ввести число заново без потери попытки).
6	Модифицируйте игру: после каждой попытки выводите оставшееся количество попыток. Например, «Осталось попыток: 2». Если пользователь угадал, показать

	«Поздравляем! Вы угадали с X попытки».
7	Реализуйте игру, где загадывается число от 1 до 100, количество попыток не ограничено. При угадывании выводите количество попыток. Добавьте возможность сдаться (ввести 0 или специальное слово «сдаться»).
8	Создайте калькулятор с помощью switch. Пользователь вводит два числа и оператор в виде строки («+», «-», «*», «/»). Выполните действие. Реализуйте обработку деления на ноль.
9	Напишите скрипт, который определяет, является ли год високосным. Условия: год делится на 4, но не делится на 100, или делится на 400. Запросите год, выведите «високосный» или «не високосный».
10	Используя тернарный оператор, присвойте переменной status значение «совершеннолетний», если возраст >= 18, и «несовершеннолетний» в противном случае. Выведите статус. Сравните с реализацией через if/else.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) HTML и JS код игры.
3) Скриншоты нескольких игровых сессий (успех, неудача, подсказки).
4) Скриншот консоли при ошибках ввода.
5) Ответы на вопросы.
6) Вывод о применении условных операторов.

Вопросы для защиты работы

1. В чём отличие if/else от switch? Когда что лучше использовать?
2. Как работают логические операторы && и ||?
3. Что такое «ложные» значения в JS? Назовите их.
4. Как сгенерировать случайное целое число от min до max?
5. Как выполнить код, если условие ложно, без else?
6. Можно ли использовать несколько условий в одном if? Пример.
7. Что такое тернарный оператор ? :? Как он записывается?

8. Как сравнить строки без учёта регистра?
9. Что произойдёт, если в switch не написать break?
10. Как организовать цикл (для повторения попыток) без использования циклов? (например, счётчик попыток)

Критерии оценивания

Параметр	Баллы
Генерация случайного числа	1
Корректная работа условий (if/else или switch)	3
Реализация подсказок и ограничения попыток	3
Обработка нечислового ввода	2
Оформление отчета	1

Лабораторная работа № 9

Циклы и функции в JS

Цель: Функции как способ организации кода. Циклы for/while для генерации контента.

Теоретическое обоснование

Функции – блоки кода, которые можно вызывать многократно.

```
js

// Function declaration (всплывает)
function sum(a, b) {
  return a + b;
}

// Function expression
const multiply = function(a, b) { return a * b; };

// Стрелочная функция (современная)
const divide = (a, b) => a / b;
```

- Параметры – переменные, перечисленные при объявлении. Аргументы – значения, передаваемые при вызове.
- return – возвращает значение. Без return функция возвращает undefined.

Циклы:

- for (инициализация; условие; шаг) { } – используется, когда известно количество итераций.
- while (условие) { } – выполняется, пока условие истинно.
- do { } while (условие) – сначала выполнит тело, потом проверит.
- for (let item of array) { } – для перебора элементов массива.
- for (let key in object) { } – для перебора ключей объекта (с осторожностью).

Пример генерации HTML-списка:

```
js

function renderList(items) {
  let html = '<ul>';
  for (let i = 0; i < items.length; i++) {
    html += '<li>' + items[i] + '</li>';
  }
  html += '</ul>';
  return html;
}
document.getElementById('list-container').innerHTML = renderList(['яблоко', 'банан']);
```

Циклы и DOM – часто используются для создания множества элементов,

заполнения таблиц, рендеринга данных.

Область видимости – переменные, объявленные внутри функции, недоступны снаружи. let и const имеют блочную видимость ({}).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Напишите функцию greet(name), которая возвращает строку «Привет, name!». Вызовите её с тремя разными именами и выведите результаты в консоль.
2	Создайте массив fruits = ["яблоко", "банан", "апельсин", "киви"]. Используя цикл for, выведите каждый элемент массива в консоль в формате «Фрукт: ...». Затем сделайте то же с циклом forEach.
3	Напишите функцию generateList(items), которая принимает массив строк и возвращает строку с HTML-разметкой ... . Например, для массива ["a","b"] вернёт - a - b . Проверьте, что строка корректна.
4	В HTML-странице создайте контейнер . Используя результат работы функции generateList, вставьте сгенерированный HTML в контейнер через innerHTML. Отобразите список на странице.
5	Модифицируйте функцию generateList, добавив второй параметр ordered (по умолчанию false). Если ordered === true, функция должна возвращать нумерованный список , иначе – маркированный .
6	Используя цикл while, выведите числа от 10 до 1 в обратном порядке (каждое число в консоль). Затем с помощью цикла for выведите только чётные числа от 2 до 20.
7	Создайте функцию sumRange(a, b), которая возвращает сумму всех целых чисел от a до b включительно. Предполагается, что a <= b. Проверьте работу для (1,5) → 15, для (3,7) → 25.
8	Напишите функцию filterEven(numbers), которая принимает массив чисел и возвращает новый массив, содержащий только чётные числа. Используйте

	метод filter или цикл. Проверьте: filterEven([1,2,3,4,5,6]) → [2,4,6].
9	Создайте HTML-страницу: текстовое поле (<input type="text">) и кнопка «Добавить». При нажатии на кнопку текст из поля добавляется в массив items. Затем вызывается функция renderList, которая полностью перерисовывает список на основе массива items (используя generateList). Реализуйте также возможность удаления элементов (по клику на элемент или кнопку «Удалить» рядом).
10	Реализуйте функцию renderTable(rows, cols), которая генерирует HTML-таблицу rows строк и cols столбцов. В каждой ячейке должно быть число – (номер строки, номер столбца), например, «1,1» для первой ячейки. Вставьте таблицу в div. Вызовите функцию для 5 строк и 4 столбцов.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) Код функции генерации списка и вызова.
3) Скриншот страницы с динамически созданным списком.
4) Скриншот консоли с работой циклов и функций.
5) Ответы на вопросы.
6) Вывод о преимуществах функций.

Вопросы для защиты работы

1. Как объявить функцию в JavaScript? Чем отличается function declaration от function expression?
2. Что такое параметры и аргументы функции?
3. Как вернуть значение из функции?
4. В чём разница между циклами for и while?
5. Как прервать цикл досрочно? (break, continue)
6. Как перебрать массив с помощью for?
7. Можно ли вложить цикл в цикл? Приведите пример.
8. Что такое область видимости переменной (scope) в функциях?

9. Как динамически создать HTML-элемент через JS?
10. Чем `innerHTML` отличается от `textContent`?

Критерии оценивания

Параметр	Баллы
Правильное создание функции с параметрами	2
Использование циклов для обработки массива	2
Генерация корректного HTML-списка	3
Дополнительные функции (фильтрация, сумма)	2
Оформление отчёта	1

Лабораторная работа № 10

Массивы и объекты в JS

Цель: Создание, перебор, добавление/удаление элементов. Объекты как модель данных.

Теоретическое обоснование

Массивы – упорядоченные коллекции. Создание:

```
js

let arr = [1, 2, 3];
let empty = new Array(5); // массив длины 5 (пустые слоты)
```

Основные методы:

- push(el) – добавить в конец.
- pop() – удалить с конца.
- unshift(el) – добавить в начало.
- shift() – удалить с начала.
- splice(index, deleteCount, ...items) – удалить/заменить/добавить.
- slice(start, end) – копия части массива.
- concat(arr2) – объединение.
- indexOf(el), includes(el) – поиск.
- forEach(callback) – перебор.
- map(callback) – создание нового массива на основе преобразования.
- filter(callback) – фильтрация.
- reduce(callback, initial) – свертка.

Объекты – неупорядоченные коллекции пар «ключ-значение»:

```
js

let user = {
  name: "Иван",
  age: 30,
  "likes cats": true   // ключ с пробелом в кавычках
};
// Доступ:
user.name;           // "Иван"
user["likes cats"];  // true
```

Можно добавлять/удалять свойства динамически: user.email = "ivan@example.com"; delete user.age;.

Массив объектов – типичная модель данных для Todo-списка:

```
js

let tasks = [
  { id: 1, text: "Купить молоко", completed: false },
  { id: 2, text: "Сделать домашку", completed: true }
];
```

Перебор массива объектов:

```
js

tasks.forEach(task => {
  console.log(task.text);
});
```

Отображение на странице – функция рендеринга, которая создаёт DOM-элементы для каждого объекта и добавляет их в контейнер. При изменении массива (добавление, удаление, изменение) нужно вызывать рендеринг заново (или обновлять точечно).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте массив colors = ["red", "green", "blue"]. Добавьте в конец «yellow» (метод push). Удалите первый элемент (метод shift). Выведите получившийся массив в консоль.
2	Создайте объект book со свойствами: title (строка), author (строка), year (число), isRead (булево) . Используя цикл for...in, выведите все свойства и их значения в консоль.
3	Создайте массив из трёх объектов task с полями: id (число), text (строка), completed (булево). Например: {id: 1, text: "Купить хлеб", completed: false}. Выведите каждый объект в консоль в формате «Задача: текст, выполнена: да/нет».
4	Напишите функцию displayTasks(tasks), которая принимает массив задач и выводит их на страницу в виде списка . Каждая задача должна отображаться как с текстом задачи. Если задача выполнена (completed: true), текст должен быть перечёркнут (стиль text-decoration: line-through). Используйте цикл для создания элементов.
5	Добавьте на страницу возможность отмечать задачу выполненной:

	при клике на текст задачи меняйте свойство completed на противоположное и обновляйте отображение (перерисовывайте список). Подсказка: используйте делегирование событий.
6	Реализуйте функцию addTask(text), которая создаёт новый объект задачи (id генерируется как Date.now() или счётчик), добавляет его в массив и вызывает перерисовку. На странице добавьте текстовое поле и кнопку «Добавить».
7	Реализуйте функцию deleteTask(id), которая удаляет задачу из массива по id (метод filter). Рядом с каждой задачей в списке добавьте кнопку «Удалить». При клике на кнопку вызывайте удаление и перерисовку.
8	Создайте массив чисел [5, 12, 8, 130, 44]. Найдите максимальное и минимальное значение с помощью цикла (без встроенных Math.max/Math.min). Затем сделайте то же с помощью методов reduce или Math.max.apply.
9	Создайте массив объектов «пользователи»: [{name: "Анна", age: 25}, {name: "Иван", age: 30}, {name: "Мария", age: 22}]. Отсортируйте массив по возрасту (по возрастанию) с помощью метода sort. Выведите отсортированный массив в виде таблицы HTML (используйте цикл для создания строк таблицы).
10	Реализуйте полный Todo-список (как в предыдущих заданиях) со следующими возможностями: • Добавление задачи, • Удаление задачи, • Отметка о выполнении, • Фильтр: показывать все / только выполненные / только активные. Для фильтра используйте метод filter и перерисовку. Данные храните в массиве.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) Код с массивом задач и функциями.
3) Скриншот отображения списка задач.
4) Скриншот после добавления/удаления задач.
5) Ответы на вопросы.

6) Вывод о структурах данных.

Вопросы для защиты работы

1. Как создать пустой массив? Как добавить элемент в конец и в начало?
2. Как удалить элемент по индексу?
3. Чем отличается for...of от for...in?
4. Как перебрать массив объектов и вывести определённое поле?
5. Что такое деструктуризация объекта?
6. Как скопировать массив (поверхностно)?
7. Как проверить, является ли переменная массивом?
8. Какие методы массива изменяют исходный массив, а какие возвращают новый?
9. Как создать объект с динамическими ключами?
10. Что такое JSON и как преобразовать объект в JSON и обратно?

Критерии оценивания

Параметр	Баллы
Создание массива объектов	2
Функции добавления, удаления, отображения	4
Использование методов перебора (forEach, filter)	2
Ответы на вопросы	1
Оформление отчёта	1

Лабораторная работа № 11

Работа с DOM (Document Object Model)

Цель: Поиск элементов на странице. Изменение контента и стилей через JS. Реализация динамического изменения темы страницы (светлая/темная) по кнопке.

Теоретическое обоснование

DOM – это объектное представление HTML-документа. Браузер строит дерево узлов, где каждый элемент, текст, атрибут – это узел. JavaScript может изменять это дерево.

Поиск элементов:

- document.getElementById("id") – один элемент.
- document.querySelector("css selector") – первый подходящий.
- document.querySelectorAll("css selector") – коллекция всех (псевдомассив).
- document.getElementsByClassName, getElementsByTagName – устаревающие, но работают.

Изменение содержимого:

- element.textContent = "новый текст" – безопасно, не интерпретирует HTML.
- element.innerHTML = "жирный" – вставляет HTML (опасно при вводе пользователя – XSS).
- element.innerText – похож на textContent, но учитывает стили (медленнее).

Изменение стилей:

- element.style.color = "red"; – инлайн-стиль.
- element.style.backgroundColor = "black"; (camelCase).
- Лучше управлять классами: element.classList.add("dark"), remove("dark"), toggle("dark").

Создание и удаление элементов:

```
js

let newDiv = document.createElement("div");
newDiv.textContent = "Привет";
document.body.appendChild(newDiv);

let oldDiv = document.getElementById("old");
oldDiv.remove(); // или parent.removeChild(child)
```

Смена темы – типичная задача:

- В CSS определён класс .dark-theme для body.
- При клике на кнопку: document.body.classList.toggle("dark-theme").
- Тему можно сохранить в localStorage.

Важно: манипуляции с DOM могут быть дорогими. Для частых обновлений используют DocumentFragment или перерисовку только изменённых элементов.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте страницу с абзацем (Исходный текст) и кнопкой «Изменить». По нажатию на кнопку измените текст абзаца на «Новый текст, изменённый через JS».
2	На странице создайте несколько элементов с классом item (например, три абзаца или три). Используя querySelectorAll, найдите все элементы с классом item и установите им красный цвет текста.
3	Реализуйте переключение светлой/тёмной темы: • В CSS определите класс .dark-theme для body (тёмный фон, светлый текст). • Добавьте кнопку «Сменить тему». • По клику на кнопку добавляйте/удаляйте класс .dark-theme у body с помощью classList.toggle. • Проверьте, что тема меняется.
4	Создайте текстовое поле <input type="text" id="nameInput"> и заголовок <h2 id="displayName">. При вводе текста в поле (событие input) дублируйте введённый текст в заголовок. То есть заголовок динамически обновляется по мере печати.
5	Разместите на странице три изображения (теги <img> с реальными картинками) и две кнопки: «Скрыть картинки» и «Показать картинки». При нажатии на «Скрыть» установите для всех изображений display: none, при нажатии на «Показать» – display: inline-block (или block).
6	На основе предыдущего Todo-списка (ЛР10) измените добавление задач: теперь при добавлении задачи вы

	должны динамически создавать DOM-элемент (document.createElement), а не перерисовывать весь список. При удалении задачи удаляйте соответствующий элемент из DOM. При этом массив задач всё равно нужно поддерживать в актуальном состоянии.
7	Создайте кнопку «Случайный цвет». При её нажатии цвет фона страницы (document.body.style.backgroundColor) меняется на случайный цвет. Случайный цвет можно получить, сгенерировав три числа от 0 до 255 и сформировав строку rgb(r, g, b).
8	Найдите на странице все ссылки (<a>). Пройдитесь по ним циклом и замените их текст на «Ссылка заблокирована», а атрибут href измените на #. Это делается для демонстрации манипуляции атрибутами.
9	Создайте контейнер и кнопку «Добавить блок». При каждом клике добавляйте в контейнер новый с текстом «Новый блок» и случайным цветом фона. Также добавьте возможность удалять блок по клику на него.
10	Реализуйте переключение тем с сохранением выбранной темы в переменную (но не в localStorage). Добавьте индикатор текущей темы (например, надпись «Текущая тема: светлая»). При переключении обновляйте надпись.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) HTML/CSS/JS код переключения темы.
3) Скриншоты светлой и тёмной темы.
4) Скриншот с изменением текста/стилей.
5) Ответы на вопросы.
6) Вывод о манипуляции DOM.

Вопросы для защиты работы

1. Что такое DOM?
2. Как найти элемент по id, по классу, по селектору CSS?
3. В чём разница между `querySelector` и `querySelectorAll`?
4. Как изменить внутренний HTML элемента? Чем это опасно?
5. Как изменить CSS-стиль элемента через JS?
6. Как добавить или удалить класс у элемента?
7. Как создать новый элемент и добавить его в DOM?
8. Как удалить элемент из DOM?
9. Что такое `event` и как его обработать? (базово)
10. Как получить значение из текстового поля `<input>`?

Критерии оценивания

Параметр	Баллы
Поиск элементов и изменение контента	2
Реализация смены темы через класс	3
Динамическое создание/удаление элементов	2
События (кнопки, ввод)	2
Ответы на вопросы	1

Лабораторная работа № 12

Обработка событий в JS

Цель: Вешаем обработчики на клики, ввод текста. Основы работы с формами.

Теоретическое обоснование

Событие – сигнал от браузера о том, что что-то произошло (клик, наведение, загрузка, нажатие клавиши и т.д.).

Способы назначения обработчиков:

1. Атрибут HTML: <button onclick="alert('Click')"> – не рекомендуется (смешивает логику и разметку).
2. Свойство DOM-объекта: button.onclick = function() { ... } – можно назначить только один обработчик.
3. addEventListener – предпочтительный способ: element.addEventListener('click', handler). Можно добавить несколько обработчиков на одно событие, а также удалить removeEventListener.

Объект события – передаётся в обработчик первым аргументом:

```
js


button.addEventListener('click', function(event) {
  console.log(event.target); // элемент, на котором произошёл клик
  console.log(event.type);   // "click"
});
```

Предотвращение действий по умолчанию:

- event.preventDefault() – отменяет отправку формы, переход по ссылке и т.д.

Всплытие событий – событие сначала срабатывает на самом вложенном элементе, затем поднимается к родителям. Можно остановить: event.stopPropagation().

Делегирование событий – вешаем обработчик на родителя, а внутри проверяем event.target. Полезно для динамически добавляемых элементов.

Формы – событие submit на форме. Чтобы получить данные:

```
js

form.addEventListener('submit', (e) => {
  e.preventDefault();
  let input = document.getElementById('task-input');
  let text = input.value;
  addTask(text);
  input.value = '';
});
```

Динамическое добавление задач – при клике на кнопку «Добавить» читаем текст из поля, создаём элемент списка, вставляем в DOM. Для удаления каждой задачи – вешаем обработчик на кнопку «Удалить» внутри создаваемого элемента.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Установите VS Code (официальный сайт) и браузер Google Chrome (или Firefox). Создайте папку lab1 на рабочем столе. В VS Code откройте эту папку, создайте файл index.html. Напишите базовую структуру HTML-документа, внутри <body> разместите фразу «Hello, World!». Сохраните файл. Откройте его в браузере двойным щелчком. Убедитесь, что надпись отображается.
2	Откройте созданную страницу в браузере, вызовите DevTools (F12 или Ctrl+Shift+I). Перейдите на вкладку Elements. Найдите узел <body> или <h1> (если использовали). Дважды кликните по тексту «Hello, World!» и измените его на «Мой первый HTML». Изменение должно сразу отобразиться на странице. Сделайте скриншот.
3	Дополните HTML-файл из задания 1: в секции <head> добавьте метатег <meta charset="UTF-8"> и тег <title> с текстом «Страница студента [Ваша фамилия и имя]». Проверьте, что вкладка браузера отображает правильный заголовок.
4	Создайте страницу, которая в <body> содержит три разных приветствия на русском, английском и немецком языках (например, «Привет!», «Hello!», «Hallo!»). Каждое приветствие должно быть оформлено в отдельном абзаце <p>. Добавьте горизонтальную линию <hr> между ними.

5	Используя DevTools (вкладка Elements → справа вкладка Computed или просто наведение на элемент), измерьте размеры (ширину и высоту) элемента <body> на вашей странице. Также определите размеры видимой области окна (viewport) – это можно увидеть вверху страницы при включённом режиме адаптивного дизайна (Toggle device toolbar). Запишите значения.
6	В VS Code установите следующие расширения: • Russian Language Pack (для русификации интерфейса), • Live Server (для автоматического обновления страницы), • Prettier (для форматирования кода). После установки перезапустите VS Code. Откройте ваш index.html, щёлкните правой кнопкой мыши по файлу и выберите «Open with Live Server». Убедитесь, что страница открылась в браузере и при изменении текста в HTML автоматически обновляется.
7	Создайте HTML-файл, в котором с помощью HTML-комментариев (<!-- ... -->) объясните назначение следующих секций: <!DOCTYPE html>, <html>, <head>, <body>. Например: <!-- DOCTYPE сообщает браузеру версию HTML -->. Сохраните файл как structure.html.
8	Создайте страницу, где в <head> задан базовый URL: <base href="https://example.com/">. В теле страницы разместите ссылку Фото. При клике по ссылке браузер должен перейти по адресу https://example.com/images/photo.jpg (это не обязательно рабочий URL, просто проверка работы base). Откройте DevTools, наведите на ссылку – внизу браузера должен отобразиться полный URL.
9	Откройте вашу страницу «Hello, World!» в браузере, откройте DevTools, перейдите на вкладку Network. Обновите страницу (F5). Найдите в списке файл index.html. Запишите время загрузки (поле Time). Также найдите вкладку Performance и нажмите запись (record) – после остановки записи посмотрите, сколько времени заняла отрисовка страницы.

10	Создайте две страницы: page1.html и page2.html. На первой странице разместите ссылку на вторую, на второй – ссылку обратно на первую. Проверьте переходы. Обе страницы должны иметь заголовки (тег <title>), соответствующие их содержанию.

Содержание отчета и его форма

- Титульный лист с названием ЛР, ФИО, вариант.
- Код формы и списка задач с обработчиками.
- Скриншоты добавления, удаления, фильтрации задач.
- Скриншот консоли при сабмите формы.
- Ответы на вопросы.
- Вывод о работе событий.

Вопросы для защиты работы

1. Чем отличается `onclick` от `addEventListener`?
2. Как предотвратить стандартное действие элемента (например, отправку формы)?
3. Как получить данные из формы после её отправки?
4. Что такое объект события `e`?
5. Как удалить обработчик события?
6. Как добавить обработчик на несколько элементов сразу (цикл)?
7. Как отследить изменение текстового поля?
8. Как работает делегирование событий? (базово)
9. Как динамически добавленным элементам назначить обработчик?
10. Что такое всплытие событий и как его остановить?

Критерии оценивания

Параметр	Баллы
Назначение обработчиков через addEventListener	2
Добавление задачи через форму (без перезагрузки)	3

Удаление и отметка выполнения	2
Фильтрация или поиск	2
Оформление отчёта	1

Лабораторная работа № 13

Хранение данных на клиенте

Цель: LocalStorage и SessionStorage. Сохранение состояния приложения между сессиями. Модификация "Списка задач": сохранение списка в localStorage и загрузка при обновлении страницы.

Теоретическое обоснование

Web Storage – механизм для хранения строковых данных в браузере. Доступен через объекты localStorage и sessionStorage.

LocalStorage:

- Данные сохраняются после закрытия браузера.
- Объём – обычно 5–10 МБ.
- Живут до тех пор, пока не будут очищены явно или пользователем.

SessionStorage:

- Данные живут только в рамках текущей вкладки/сессии.
- При закрытии вкладки удаляются.

Основные методы:

```
js


localStorage.setItem('key', 'value');
let value = localStorage.getItem('key');
localStorage.removeItem('key');
localStorage.clear();
```

Сохранение объектов и массивов – хранилище работает только со строками. Используем JSON:

```
js


let tasks = [{id:1, text:"купить"}];
localStorage.setItem('tasks', JSON.stringify(tasks));
let restored = JSON.parse(localStorage.getItem('tasks'));
```

Загрузка состояния приложения – при загрузке страницы проверяем, есть ли сохранённые данные. Если есть – парсим и рендерим их. Если нет – создаём пустой массив или загружаем по умолчанию.

Сохранение при изменениях – после каждого добавления, удаления, изменения задачи сериализуем массив и сохраняем в localStorage.

Очистка – кнопка «Очистить» вызывает localStorage.removeItem('tasks') и очищает список на странице.

Практический смысл – приложение «Список задач» не теряет данные после перезагрузки страницы, что имитирует поведение настоящего веб-приложения.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Сохраните в localStorage строку «Привет» под ключом greeting. Затем извлеките её и выведите в консоль. Удалите ключ greeting и убедитесь, что он пропал.
2	Создайте объект пользователя {name: "Иван", age: 30}. Сохраните его в localStorage (преобразовав в JSON). Прочитайте, преобразуйте обратно в объект и выведите имя в консоль.
3	Модифицируйте Todo-приложение: при каждом добавлении, удалении, изменении задачи сохраняйте весь массив задач в localStorage (ключ tasks). Используйте JSON.stringify.
4	При загрузке страницы проверяйте, есть ли в localStorage ключ tasks. Если есть – загружайте задачи оттуда и отображайте их. Если нет – создавайте пустой массив или несколько задач по умолчанию.
5	Добавьте кнопку «Очистить историю». При её нажатии удаляйте данные из localStorage (localStorage.removeItem('tasks')) и очищайте список задач на странице.
6	Реализуйте счётчик посещений страницы. При каждой загрузке (событие load или просто скрипт внизу) увеличивайте значение в localStorage по ключу visits на 1. Выведите сообщение: «Вы посетили страницу N раз».
7	Сохраняйте выбранную тему страницы (светлая/тёмная) в localStorage по ключу theme. При загрузке страницы

	читайте эту настройку и применяйте соответствующую тему. При переключении темы обновляйте localStorage.
8	Используйте sessionStorage для хранения состояния фильтра задач (например, выбранный фильтр «активные»). Проверьте, что после закрытия вкладки и повторного открытия фильтр сбрасывается в «все».
9	Создайте приложение «Заметки»: текстовое поле и кнопка «Сохранить заметку». Заметки сохраняйте в localStorage в виде массива. Отображайте все сохранённые заметки. Добавьте возможность удалять заметки.
10	Реализуйте экспорт/импорт списка задач: • Кнопка «Сохранить в файл» – сериализуйте массив задач в JSON и предложите скачать как файл tasks.json. • Кнопка «Загрузить из файла» – используйте <input type="file"> для выбора JSON-файла, прочитайте его и замените текущий список задач на загруженный (сохранив в localStorage).

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) Код модифицированного Todo-приложения с сохранением.
3) Скриншот до перезагрузки страницы и после (задачи восстановлены).
4) Скриншот работы кнопки очистки.
5) Ответы на вопросы.
6) Вывод о необходимости клиентского хранения.

Вопросы для защиты работы

1. В чём разница между `localStorage` и `sessionStorage`?
2. Какой объём данных можно хранить в `localStorage`?
3. Как сохранить объект в `localStorage`?
4. Как удалить конкретный элемент из `localStorage`?
5. Как очистить всё хранилище?
6. Что произойдёт, если попытаться сохранить объект без `JSON.stringify()`?
7. Как обработать ситуацию, когда данных в `localStorage` нет?

8. Можно ли хранить функции в localStorage?
9. Где физически хранятся данные localStorage в браузере?
10. Как прослушать событие изменения localStorage?

Критерии оценивания

Параметр	Баллы
Сохранение массива задач в localStorage	3
Загрузка и отображение при старте	3
Сохранение темы/фильтра	2
Ответы на вопросы	1
Оформление отчёта	1

Лабораторная работа № 14

Основы работы с формами и валидация

Цель: Типы полей ввода. Проверка данных на стороне клиента с помощью JS. Форма регистрации с проверкой пароля и email на корректность.

Теоретическое обоснование

Типы полей ввода HTML5:

- text, password, email, tel, number, date, checkbox, radio, select, textarea.

Атрибуты валидации HTML5:

- required – обязательное поле.
- minlength, maxlength – длина текста.
- min, max – для чисел.
- pattern – регулярное выражение.

Но клиентская валидация на JS даёт больше контроля и единообразие.

Проверка email – простая регулярка: /.+@.+\..+/ (есть @ и точка после). Более строгая: /^[^\s@]+@([^\s@]+\.)+[^\s@]+$/.

Проверка пароля:

- Длина не менее 6 символов.
- Содержит хотя бы одну цифру: /\d/.

- Содержит заглавную букву: /[A-Z]/.

Подтверждение пароля – сравнение значения password и password_confirm.

Моментальная валидация – события input или change. При вводе проверяем поле и показываем сообщение об ошибке рядом.

Блокировка кнопки отправки – атрибут disabled. Устанавливаем true, если форма невалидна.

Отображение ошибок – создаём динамически или показываем/скрываем заранее созданные блоки.

Пример валидации перед отправкой:

```
js

form.addEventListener('submit', (e) => {
  if (!isFormValid()) {
    e.preventDefault();
    showErrors();
  }
});
```

Важно: клиентская валидация – только для удобства. Серверная валидация обязательна для безопасности.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Создайте форму регистрации с полями: имя (только буквы, обязательно), email (обязательный, формат email). При отправке проверяйте поля, показывайте ошибки под ними. Если ошибок нет – выводите alert("Успех").
2	Добавьте поле пароля. Требования: длина не менее 6 символов, содержит хотя бы одну цифру. При невыполнении показывайте сообщение.
3	Добавьте поле «Подтверждение пароля». Проверьте совпадение с паролем. Если не совпадает – ошибка.
4	Реализуйте моментальную валидацию (событие input): при вводе в поле сразу показывайте зелёную галочку (✓) или красный крест (✗) рядом с полем. Используйте иконки или текст.

5	Заблокируйте кнопку отправки, пока форма не валидна. Для этого в обработчиках input проверяйте все поля и устанавливайте атрибут disabled кнопки в true/false.
6	Добавьте поле «Дата рождения» (тип date). При отправке проверьте, что пользователю не менее 18 лет (вычислите возраст). Если младше – ошибка.
7	Создайте поле выбора страны (<select>) и поле «Город» (<input>). Сделайте так, что если выбрана страна «Россия», поле «Город» становится обязательным; для других стран – необязательным. При изменении страны динамически меняйте атрибут required поля города.
8	Реализуйте проверку уникальности логина (имитация). Создайте массив существующих логинов ["admin", "user123", "anna"]. При вводе логина проверяйте, нет ли его в массиве. Если есть – показывайте ошибку «Логин занят».
9	Валидация с отображением ошибок под каждым полем. Сообщения об ошибках должны появляться динамически. При успешной отправке показывайте alert("Регистрация успешна") и очищайте форму.
10	Добавьте чекбокс «Согласие на обработку данных». Кнопка отправки должна быть активна только если чекбокс отмечен. Покажите пример стилизации невалидных полей (красная рамка) с помощью CSS-псевдоклассов :invalid, :valid.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) Код формы регистрации с валидацией.
3) Скриншоты ошибок при вводе (пустое поле, неверный email, короткий пароль).
4) Скриншот успешной отправки.
5) Ответы на вопросы.
6) Вывод о важности валидации.

Вопросы для защиты работы

1. Какие атрибуты HTML5 помогают валидации (required, pattern, type)?

2. Как отключить встроенную браузерную валидацию?
3. Как проверить email с помощью регулярного выражения (простейший вариант)?
4. Как получить доступ к значению поля формы?
5. Как программно показать ошибку рядом с полем?
6. Как проверить, что в поле введено число, а не текст?
7. Что такое «моментальная валидация»?
8. Как предотвратить отправку формы при невалидных данных?
9. Как стилизовать валидные и невалидные поля с помощью CSS-псевдоклассов `:valid`, `:invalid`?
10. Почему валидация на клиенте не заменяет серверную?

Критерии оценивания

Параметр	Баллы
Проверка всех полей (имя, email, пароль, подтверждение)	3
Отображение ошибок в реальном времени	2
Блокировка кнопки отправки	1
Дополнительные проверки (возраст, уникальность)	3
Оформление отчёта	1

Лабораторная работа № 15

Асинхронность в JS (введение)

Цель: Коллбэки и промисы. Понятие API. Выполнение учебных запросов к тестовым API (jsonplaceholder).

Теоретическое обоснование

Асинхронность – выполнение операций без блокировки основного потока. JS однопоточный, но использует event loop.

Коллбэки (callbacks) – функции, переданные в другие функции и вызываемые по завершении операции. Пример с setTimeout:

```
js

setTimeout(() => {
  console.log('Прошло 2 секунды');
}, 2000);
```

Проблема коллбэков – «ад коллбэков» (множественная вложенность).

Промисы (Promise) – объект, представляющий результат асинхронной операции. Состояния:

- pending – ожидание.
- fulfilled – успешно завершён.
- rejected – ошибка.

Создание промиса:

```
js

const promise = new Promise((resolve, reject) => {
  // асинхронный код
  if (успех) resolve(результат);
  else reject(ошибка);
});
```

Обработка:

```
js

promise
  .then(result => console.log(result))
  .catch(error => console.error(error))
  .finally(() => console.log('Завершено'));
```

API (Application Programming Interface) – набор правил и эндпоинтов для обмена данными. Тестовое API JSONPlaceholder: https://jsonplaceholder.typicode.com/.

Fetch API – современный способ делать HTTP-запросы. Возвращает промис:

```js
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    if (!response.ok) throw new Error('Ошибка сети');
    return response.json();
  })
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Async/await – синтаксический сахар над промисами:

```js
async function loadData() {
  try {
    let response = await fetch(url);
    let data = await response.json();
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}
```

Promise.all – дожидается выполнения всех промисов из массива. Если один отклонён, то результат отклонён.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Напишите функцию loadData(callback), которая имитирует загрузку данных через setTimeout (задержка 2 секунды). Функция должна вызвать callback с переданным аргументом «Данные загружены». Вызовите функцию и выведите результат.
2	Создайте промис, который разрешается через 1 секунду со значением «Успех». Обработайте его с помощью .then(), выведите результат в консоль.
3	Создайте промис, который случайным образом (Math.random() > 0.5) разрешается или отклоняется. В .then выведите «Успех», в .catch – «Ошибка».
4	Используя fetch, сделайте GET-запрос к https://jsonplaceholder.typicode.com/posts/1. Полученный ответ преобразуйте в JSON и выведите в консоль.
5	Выполните запрос к https://jsonplaceholder.typicode.com/users, получите список пользователей. Отобразите их имена в виде маркированного списка на странице.

6	Намеренно измените URL на несуществующий (например, https://jsonplaceholder.typicode.com/posts/99999). Обработайте ошибку с помощью .catch() и покажите пользователю сообщение «Данные не найдены».
7	Цепочка промисов: сначала получите пост с id=1 (/posts/1), затем из результата возьмите userId и получите данные пользователя (/users/${userId}). Выведите в консоль имя пользователя и заголовок поста.
8	Используйте Promise.all для параллельной загрузки трёх постов: id=1,2,3. Дождитесь загрузки всех и выведите их в консоль (массив из трёх объектов).
9	Создайте функцию delay(ms), которая возвращает промис, разрешающийся через ms миллисекунд. Напишите асинхронную функцию, которая вызывает delay(2000), затем выводит «Прошло 2 секунды».
10	Перепишите любое из предыдущих заданий (например, задание 5) с использованием async/await вместо .then(). Оберните в try/catch для обработки ошибок.

Содержание отчета и его форма

1) Титульный лист с названием ЛР, ФИО, вариант.
2) Код запросов к JSONPlaceholder.
3) Скриншот консоли с выведенными данными.
4) Скриншот страницы с отображением пользователей.
5) Ответы на вопросы.
6) Вывод о работе с асинхронностью..

Вопросы для защиты работы

1. Что такое асинхронность в JS и зачем она нужна?
2. Что такое callback hell и как с ним бороться?
3. Что такое Promise? Какие у него состояния?
4. Чем отличается .then() от async/await?
5. Как обработать ошибку в async/await?
6. Что делает fetch() и что он возвращает?

7. Как извлечь JSON из ответа `fetch`?
8. Как отправить POST-запрос с помощью `fetch`?
9. Что такое `Promise.all` и когда его используют?
10. Что такое API? Приведите примеры.

Критерии оценивания

Параметр	Баллы
Понимание коллбэков и промисов (простой пример)	2
Выполнение GET-запроса и отображение данных	3
Обработка ошибок	2
Использование Promise.all или async/await	2
Оформление отчёта	1

Лабораторная работа № 16

Работа с удаленными данными (Fetch API)

Цель: GET-запросы. Отображение списка постов/пользователей с сервера на странице. Замена локального массива задач на загрузку задач с внешнего API.

Теоретическое обоснование

GET-запросы – получение данных. Пример списка постов:

```
js

fetch('https://jsonplaceholder.typicode.com/posts')
  .then(res => res.json())
  .then(posts => {
    posts.forEach(post => {
      // отобразить на странице
    });
  });
```

POST-запрос – отправка новых данных:

```
js

fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ title: 'Новый пост', body: 'текст', userId: 1 })
})
.then(res => res.json())
.then(newPost => console.log(newPost));
```

Замена локального массива задач – вместо хранения задач в памяти, при загрузке приложения делаем fetch к API (например, /todos). Отображаем полученные задачи. При добавлении новой задачи отправляем POST (имитируя сохранение на сервере). В реальном API обычно нельзя добавлять задачи, но JSONPlaceholder возвращает имитированный ответ.

Индикатор загрузки – пока выполняется запрос, показываем спиннер или текст «Загрузка…». Скрываем после получения ответа (в .finally или после await).

Обработка ошибок – проверяем response.ok, показываем сообщение пользователю. Также ловим сетевые ошибки (нет соединения) через .catch.

Пагинация – параметры _page и _limit: https://jsonplaceholder.typicode.com/posts?_page=2&_limit=10.

Бесконечный скролл – отслеживаем прокрутку до низа страницы и подгружаем следующую страницу.

Фильтрация на клиенте – загружаем все данные один раз, затем фильтруем по поисковому запросу (без повторных запросов).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Задание
1	Загрузите с https://jsonplaceholder.typicode.com/posts список постов. Выведите на страницу заголовки постов в виде маркированного списка.
2	Добавьте кнопку «Загрузить посты». При клике на кнопку появляется индикатор загрузки (текст «Загрузка...»), затем посты отображаются. Кнопка должна быть неактивна во время загрузки.
3	Отобразите не только заголовки, но и тело каждого поста. Реализуйте аккордеон: при клике на заголовок поста показывается/скрывается тело поста.
4	Загрузите список задач (todos) с https://jsonplaceholder.typicode.com/todos. Отобразите их как чекбоксы (выполнено/не выполнено). Для выполненных задач чекбокс должен быть отмечен.
5	Модифицируйте Todo-приложение (ЛР13) так, чтобы при загрузке страницы данные сначала пытались загрузиться с API https://jsonplaceholder.typicode.com/todos?_limit=5. Если запрос не удался (ошибка сети), загружайте данные из localStorage. Если и там нет – показывайте пустой список.
6	Реализуйте добавление новой задачи через POST-запрос на https://jsonplaceholder.typicode.com/posts. При добавлении отправляйте объект {title: текст задачи, body: "описание", userId: 1}. Выведите ответ сервера в консоль. Обратите внимание, что реальные данные не сохранятся на сервере, но имитация работает.
7	Загрузите список пользователей (/users). Для каждого пользователя загрузите его посты (запрос к /posts?userId=id). Отобразите каждого пользователя с заголовками его постов (вложенные списки). Используйте Promise.all или вложенные fetch.
8	Создайте поиск по постам: поле ввода. После загрузки всех постов (один раз) при вводе текста в поле

	фильтруйте посты по заголовку (содержит подстроку) и отображайте только отфильтрованные.
9	Реализуйте пагинацию: загружайте по 10 постов, используя параметры _page и _limit. Добавьте кнопки «Вперед» и «Назад». При нажатии загружайте следующую/предыдущую страницу. Отображайте номер текущей страницы.
10	Обработайте ситуацию отсутствия интернета (отключите сеть в браузере). При ошибке запроса покажите сообщение «Не удалось загрузить данные. Проверьте соединение». Также предложите кнопку «Повторить».

Содержание отчета и его форма

- Титульный лист с названием ЛР, ФИО, вариант.
- Код загрузки и отображения постов/задач.
- Скриншот списка задач, загруженных с API.
- Скриншот работы пагинации или поиска.
- Ответы на вопросы.
- Вывод о работе с удалёнными данными.

Вопросы для защиты работы

1. Как выполнить GET-запрос с помощью fetch?
2. Как проверить, успешен ли ответ (статус 200)?
3. Как отправить POST-запрос с телом в JSON?
4. Как обработать ошибку соединения (нет интернета)?
5. Какие заголовки (headers) обычно нужны для POST?
6. Как отменить fetch-запрос (AbortController)?
7. Что такое CORS и как он влияет на запросы с фронта?
8. Как загрузить данные параллельно с разных эндпоинтов?
9. Как реализовать ленивую загрузку (бесконечный скролл)?
10. В чём отличие fetch от XMLHttpRequest?

Критерии оценивания

Параметр	Баллы
Загрузка и отображение данных с API	3
Индикатор загрузки и обработка ошибок	2
Замена локального массива на API в Todo	3
Дополнительные функции (POST, пагинация)	1
Оформление отчёта	1

Лабораторная работа № 17

Комплексная работа: сборка проекта

Цель: Итоговая верстка макета (интернет-магазин, блог или лендинг) по предоставленному техническому заданию.

Теоретическое обоснование

Этапы вёрстки по ТЗ:

1. Анализ макета (Figma, PSD, картинка). Определение сетки, цветов, шрифтов, отступов.
2. Создание файловой структуры:

```
project/
├── index.html
├── css/
│   ├── style.css
│   ├── reset.css (или normalize.css)
├── js/
│   └── main.js (пока пустой)
├── images/
└── fonts/ (если нужно)
```

3. Написание HTML-разметки с семантическими тегами.
4. Написание CSS:
   - Сброс стилей (reset/normalize).
   - Глобальные переменные (custom properties) для цветов, шрифтов.
   - Flexbox / Grid для раскладки.
   - Медиа-запросы для адаптивности.
5. Проверка Pixel Perfect (расширение для браузера).
6. Тестирование на разных устройствах (DevTools: режим мобильных устройств).

Адаптивность – используется подход mobile-first или desktop-first. Ключевые

брейкпоинты: 320px, 768px, 1024px, 1200px.

Сетка – чаще всего Grid для глобальной структуры, Flexbox для компонентов.

Изображения – оптимизация (сжатие, формат WebP, srcset для ретины).

Доступность – альтернативные тексты для img, правильная иерархия заголовков, контрастность.

Инструменты:

- Figma – измерение расстояний, экспорт.
- PerfectPixel – наложение макета на страницу.
- W3C Validator – проверка HTML/CSS.
- Lighthouse – аудит производительности и доступности.

Результат – статичная, но полностью рабочая страница, которая корректно отображается на любом экране.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Тема	Задание
1	Лендинг портфолио	Сверстайте одностраничный сайт-портфолио. Требования: • Шапка с меню (якорные ссылки). • Секция «О себе» (фото, текст). • Секция «Портфолио» – сетка из 6 карточек работ (изображение, название, описание). • Секция «Контакты» (форма обратной связи: имя, email, сообщение). • Подвал. • Адаптивность (мобильная версия: меню становится вертикальным, сетка портфолио – 1 колонка). • Использовать семантическую вёрстку, Flexbox/Grid.
2	Интернет-магазин	Сверстайте страницу каталога интернет-магазина одежды. Требования: • Шапка с логотипом, поиском, иконкой корзины.

		• Каталог – сетка карточек товаров (фото, название, цена, кнопка «В корзину»). Минимум 6 карточек. • Боковая панель с фильтром по категориям (CSS-фильтр без JS – просто ссылки). • Подвал с контактами. • Адаптив: на планшете 2 колонки, на телефоне 1 колонка, боковая панель переносится под каталог.
3	Блог о путешествиях	Сверстайте главную страницу блога. Требования: • Шапка с названием и меню. • Главный баннер с изображением. • Список статей: минимум 4 статьи (картинка, заголовок, анонс, дата). • Боковая панель: список популярных постов, категории. • Пагинация (только визуально, без JS). • Футер. • Использовать Grid для основной сетки (статьи + сайдбар).
4	Карточка компании	Сверстайте страницу компании. Требования: • Шапка с логотипом и навигацией. • Слайдер (статический, 3 картинки, без JS – просто три картинки в ряд или с переключением через radio). • Блок услуг: 3 колонки с иконкой, заголовком, текстом. • Блок команды: фото, имя, должность (4 человека). • Карта (iframe с Google Maps). • Контактная форма. • Адаптив.
5	Рецепты	Сверстайте страницу со списком рецептов. Требования: • Шапка с поиском (поле ввода без функционала). • Сетка рецептов (картинка,

		название, краткое описание, ссылка «Подробнее»). • При клике на «Подробнее» ведёт на отдельную страницу рецепта (можно отдельный HTML-файл). Страница рецепта содержит: крупное фото, ингредиенты (список), шаги приготовления (нумерованный список). • Используйте семантику.
6	Форум/сообщество	Сверстайте главную страницу форума. Требования: • Шапка с логотипом и входом. • Список тем: таблица или список с колонками: тема, автор, дата, количество ответов. • Форма создания новой темы (поля: заголовок, текст). Без JS, только вёрстка. • Боковая панель с последними комментариями. • Футер. • Адаптив: на мобильных таблица превращается в блоки.
7	Портфолио фотографа	Сверстайте галерею. Требования: • Шапка с меню. • Галерея – сетка из 9 фотографий (миниатюры). • При клике на миниатюру открывается модальное окно (CSS-only с помощью :target) с увеличенным фото. • Адаптив: 3 колонки на десктопе, 2 на планшете, 1 на телефоне. • Подвал.
8	Образовательный сайт	Сверстайте страницу курсов. Требования: • Шапка. • Список курсов (карточки с названием, длительностью, ценой). • Страница конкретного курса (отдельный HTML): программа

		(список тем), отзывы (карточки отзывов). • Боковое меню с навигацией по курсам. • Использовать Grid для раскладки.
9	Доставка еды	Сверстайте главную страницу сервиса доставки. Требования: • Шапка с поиском и корзиной. • Категории товаров (иконки с подписями). • Карточки блюд (фото, название, цена, вес, кнопка «В корзину»). • Корзина – отдельный блок (справа или выезжающая панель). Только вёрстка, без JS. • Адаптив.
10	Новостной портал	Сверстайте главную страницу новостного сайта. Требования: • Шапка с логотипом и меню (разделы новостей). • Главная новость – крупный блок с изображением, заголовком и анонсом. • Сетка остальных новостей (4–6 штук) – 2-3 колонки. • Сайдбар с рекламными баннерами (заглушки). • Футер со ссылками. • Адаптив.

Содержание отчета и его форма

- Титульный лист с названием ЛР, ФИО, вариант.
- Ссылка на GitHub Pages готового проекта.
- Скриншоты десктопной и мобильной версий.
- Исходный код (ссылка на репозиторий).
- Краткое описание использованных технологий.
- Ответы на вопросы.

- Вывод о сложностях вёрстки.

Вопросы для защиты работы

1. Какие этапы включает в себя вёрстка макета?
2. Как организовать файловую структуру проекта (CSS, images, fonts)?
3. Что такое Pixel Perfect и как его добиться?
4. Как использовать CSS-сетки (Grid/Flex) для сложных макетов?
5. Как обеспечить адаптивность для разных устройств?
6. Какие инструменты помогают при вёрстке (Figma, Pixel Perfect расширение)?
7. Что такое БЭМ и зачем он нужен?
8. Как проверить вёрстку на валидность и кроссбраузерность?
9. Какие метатеги нужны для мобильной вёрстки (viewport)?
10. Как оптимизировать загрузку изображений?

Критерии оценивания

Параметр	Баллы
Соответствие ТЗ (структура, элементы)	3
Качество вёрстки (семантика, валидность, адаптивность)	3
Организация кода (файлы, именование)	2
Ответы на вопросы	1
Оформление отчёта	1

Лабораторная работа № 18

Комплексная работа: динамика проекта

Цель: Оживление проекта: слайдеры, модальные окна, подгрузка контента. Защита готового проекта.

Теоретическое обоснование

Оживление проекта – добавление JavaScript-функционала к ранее свёрстанному макету.

Типичные динамические компоненты:

1. **Слайдер (карусель)** – переключение изображений по кнопкам или точкам. Реализация:
   - Хранение массива URL изображений.
   - Отображение текущего индекса.
   - При клике «вперёд» – увеличиваем индекс, обновляем src основного изображения.
   - Автоматическая прокрутка с setInterval (очистка при наведении).
2. **Модальное окно** – скрытое по умолчанию окно, которое появляется при клике (например, на кнопку «Подробнее»). Содержит затемнение (overlay). Закрытие по клику на крестик или на overlay.
3. **Фильтрация товаров / постов** – кнопки категорий. При клике скрываем/показываем элементы, основываясь на их data-атрибуте.
4. **Корзина интернет-магазина**:
   - Глобальный массив cart.
   - Кнопка «Добавить в корзину» – добавляет объект товара в массив, обновляет отображение корзины (количество, сумма).
   - Сохранение корзины в localStorage.
   - Страница корзины или виджет.
5. **Подгрузка контента** – кнопка «Загрузить ещё» делает fetch к API и добавляет новые элементы к существующим.
6. **Валидация форм** – как в ЛР14, но встроенная в проект.
7. **Переключение темы** – уже рассматривалось, может быть частью проекта.

Защита проекта – студент демонстрирует работающее приложение,

объясняет структуру кода, показывает ключевые функции, отвечает на вопросы преподавателя.

Требования к защите:

- Работоспособность всех функций.
- Читаемый код с комментариями.
- Отсутствие ошибок в консоли.
- Адаптивность и внешний вид соответствуют ТЗ.

Итог – полноценное одностраничное приложение (SPA-подобное) с серверной интеграцией (или имитацией), хранилищем и сложной логикой.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания по вариантам.

Вариант	Тема	Задание
1	Для лендинга портфолио	Добавьте к свёрстанному лендингу: • Модальное окно с подробным описанием проекта при клике на карточку портфолио. • Слайдер отзывов (минимум 3 отзыва, переключение по кнопкам). • Валидацию формы обратной связи (имя, email, сообщение). • Сохранение отправленных сообщений в localStorage (имитация). • Плавную прокрутку к якорям.
2	Интернет-магазин	Добавьте: • Корзину (добавление/удаление товаров, подсчёт суммы). Храните корзину в localStorage. • Фильтр товаров по цене (слайдер или два поля min/max). • Сортировку товаров (по цене, по названию). • Подсчёт количества товаров в иконке корзины (динамическое обновление).
3	Блог	Добавьте:

		• Подгрузку следующих статей по кнопке «Загрузить ещё» (JSON-файл или fetch к API). • Возможность ставить лайки постам (с сохранением в localStorage). • Поиск по статьям (фильтрация на клиенте). • Переключение между тёмной и светлой темой.
4	Карточка компании	Добавьте: • Слайдер партнёров (автоматическая прокрутка и кнопки). • Модальное окно для обратной связи с валидацией. • Динамическую карту (можно через API, но достаточно статического iframe с возможностью открыть в новой вкладке). • Кнопку «Показать телефон» (при клике появляется номер).
5	Рецепты	Добавьте: • Поиск по рецептам (фильтрация карточек по введённому тексту). • Модальное окно с полным рецептом при клике на «Подробнее» (без перехода на новую страницу). • Возможность добавлять рецепты в «Избранное» (список избранных рецептов в отдельном блоке, хранить в localStorage). • Оценку рецептов (звёздочки).
6	Форум	Добавьте: • Динамическое создание новых тем (форма на главной странице, данные сохраняются в localStorage). • Редактирование и удаление тем (только созданных пользователем). • Отображение количества комментариев (заглушка). • Сортировку тем по дате или по

		количеству ответов.
7	Портфолио фотографа	Добавьте: • Фильтр галереи по категориям (все, природа, портреты). • Слайдер для просмотра фото (lightbox) – при клике на миниатюру открывается модальное окно с возможностью листать стрелками. • Возможность добавлять фото в избранное (сердечко). • Анимацию появления картинок.
8	Образовательный сайт	Добавьте: • Регистрацию на курс через форму с валидацией (сохранение в localStorage). • Прогресс-бар прохождения модулей (имитация: пользователь отмечает пройденные модули). • Отзывы о курсе: форма добавления отзыва, отображение списка отзывов. • Тёмную тему.
9	Доставка еды	Добавьте: • Корзину с добавлением товаров, изменением количества, подсчётом суммы. • Модальное окно оформления заказа (форма: имя, адрес, телефон). Валидация. • Сохранение корзины в localStorage. • Фильтрацию товаров по категориям.
10	Новостной портал	Добавьте: • Подгрузку новостей с JSON-файла (или API) при загрузке страницы. • Сортировку новостей по дате (свежие сверху). • Кнопку «Скрыть/показать рекламный блок» (сохранение состояния в localStorage).

		• Возможность комментировать новости (простое добавление комментариев под новостью, без сервера, хранить в localStorage).

Содержание отчета и его форма

- Титульный лист с названием ЛР, ФИО, вариант.
- Финальная версия проекта на GitHub Pages.
- Скриншоты основных динамических фич (слайдер, модальное окно, корзина).
- Краткая инструкция по запуску.
- Ответы на вопросы.
- Вывод о проделанной работе.
- При защите – демонстрация кода и ответы на вопросы.

Вопросы для защиты работы

1. Как реализовать слайдер на чистом JS? Какие есть подходы?
2. Как сделать модальное окно, которое открывается/закрывается по клику?
3. Как динамически добавлять товары в корзину и обновлять интерфейс?
4. Какие события используются для слайдера (клик, touch)?
5. Как сохранить состояние корзины после перезагрузки?
6. Как реализовать бесконечную подгрузку контента (infinite scroll)?
7. Какие методы оптимизации производительности для слайдера?
8. Как передать данные между компонентами (например, из карточки в корзину)?
9. Как организовать защиту проекта: что нужно продемонстрировать?
10. Как документировать код проекта для защиты?

Критерии оценивания

Параметр	Баллы
Реализация минимум 3 динамических элементов (слайдер, модалка, корзина и т.д.)	4

Взаимодействие с хранилищем (localStorage, fetch)	2
Качество кода (читаемость, комментарии, отсутствие ошибок)	2
Ответы на вопросы	1
Оформление отчёта	1

СПСИОК ЛИТЕРАТУРЫ

1. Дакетт, Дж. HTML и CSS: Разработка и дизайн веб-сайтов / Джон Дакетт. – М. : Эксмо, 2019. – 512 с. – ISBN 978-5-699-80257-1.

2. Дакетт, Дж. JavaScript и jQuery: Интерактивная веб-разработка / Джон Дакетт. – М. : Эксмо, 2017. – 640 с. – ISBN 978-5-699-79068-7.

3. Флэнаган, Д. JavaScript. Подробное руководство / Дэвид Флэнаган. – 7-е изд. – СПб. : Питер, 2021. – 720 с. – ISBN 978-5-4461-1659-9.

4. Хоган, Б. HTML5 и CSS3: Веб-разработка по стандартам нового поколения / Брайан Хоган. – СПб. : Питер, 2014. – 304 с. – ISBN 978-5-496-00708-5.

5. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Робин Никсон. – 6-е изд. – СПб. : Питер, 2022. – 768 с. – ISBN 978-5-4461-1827-2.

6. Макфарланд, Д. CSS. Новые горизонты / Дэвид Макфарланд. – СПб. : Питер, 2020. – 608 с. – ISBN 978-5-4461-1477-9.

7. Кохран, Ш. CSS. Полный справочник / Шон Кохран. – М. : Вильямс, 2020. – 512 с. – ISBN 978-5-907144-23-7.

8. Шипилов, А. Frontend: введение в профессию / А. Шипилов. – М. : ДМК Пресс, 2021. – 320 с. – ISBN 978-5-97060-885-4.

9. Головатый, А. Frontend-разработка: практический курс / А. Головатый. – М. : Лаборатория знаний, 2022. – 280 с. – ISBN 978-5-00101-307-7.

10. Фрисби, М. JavaScript для детей: Самоучитель по программированию / Мэтт Фрисби. – М. : Манн, Иванов и Фербер, 2016. – 288 с. – ISBN 978-5-00100-452-5.

11. Закас, Н. JavaScript: Реактивное программирование / Н. Закас. – СПб. : Питер, 2019. – 416 с. – ISBN 978-5-4461-1105-1.