Лабораторная работа №1
DOM-дерево, интернационализация, LocalStorage

***Цель работы*:** добавить интерактивность ранее сверстанной web-странице с использованием JavaScript для реализации указанного в задании функционала

Порядок выполнения

1 Работа выполняется на базе лендинга, разработанного на ознакомительной практике.

2 Создать репозиторий GitHub для выполнения лабораторной работы. От ветки main создать ветку для работы над заданием.

3 В процессе работы над заданием ведите историю коммитов. История коммитов должна отображать процесс разработки приложения. Следуйте гайдлайну, согласно которому название каждого коммита должно начинаться с одного из перечисленных префиксов init:, feat:, fix:, refactor:, docs:.

4 Изучить советы по выполнению заданий и дополнительный материал.

5 Добавить на web-страницу следующую интерактивность:

- подсветка активной кнопки;
- перевод страницы на два языка;
- возможность переключения «светлой» и «темной» темы;
- сохранить выбранные пользователем настройки в LocalStorage (язык отображения страницы и светлая или тёмная тема);
- смена изображения на странице;
- добавление звука;
- новые элементы должны быть стилизованы.

6 Создать Pull Request из ветки лабораторной работы в ветку main. В описании Pull Request указать дату сдачи задания на проверку и ссылку.

Требования к верстке

- есть **не меньше пяти интерактивных элементов**, с которыми пользователи могут взаимодействовать. Изменяется внешний вид самого элемента и состояние курсора при наведении, плавные анимации;
- есть кнопка/кнопки, при клике по которой/которым меняются изображения;
- активный в данный момент интерактивный элемент выделяется стилем;
- есть кнопка Play/Pause, при клике по которой можно запустить или остановить проигрывание звука;

– внешний вид и функционал кнопки Play/Pause изменяется в зависимости от того, проигрывается ли в данный момент звук.

– есть кнопка en/ru;

– при клике по надписи ru англоязычная страница переводится на русский язык;

– при клике по надписи en русскоязычная страница переводится на английский язык;

– надписи en или ru, соответствующие текущему языку страницы, становятся активными т.е. выделяются стилем;

– после смены светлой и тёмной темы интерактивные элементы по-прежнему изменяют внешний вид при наведении и клике и при этом остаются видимыми на странице (нет ситуации с белым шрифтом на белом фоне);

– при загрузке страницы автоматически применяются настройки, выставленные пользователем в предыдущем сеансе работы.

Советы по выполнению задания

Подсветка активной кнопки

Подсветка – это просто выделение стилем: цветом, подчёркиванием, фоном и т. д. CSS умеет менять стиль при наведении, но для того, чтобы добавить выделение активному элементу и убрать его у неактивных элементов, понадобится js.

JavaScript умеет изменять стиль элемента. Например:

```
const portfolioBtn = document.querySelector('.portfolio-btn');

portfolioBtn.style.backgroundColor = '#bdae82';
portfolioBtn.style.color = '#000';
```

Данный код изменил цвет фона кнопки на золотой, а цвет шрифта на черный. Но таким образом js используют очень редко. Более распространённый вариант - в css указать класс с нужными свойствами, а в js присвоить его элементу. Например:

```
.portfolio-btn.active {
  color: #000;
  background-color: #bdae82;
}
portfolioBtn.classList.add('active');
```

Такой подход предпочтительней по нескольким причинам. Во-первых, сокращается количество кода: добавлять и удалять классы проще и быстрее, чем стили, которых может быть много. Например, активной кнопке нужно

указать изменение цвета шрифта при наведении, могут меняться и другие свойства. И для изменения каждого из этих свойств нужно будет писать js-код, а когда дизайнер внесёт правки, переписывать его. Во-вторых, соблюдается принцип разделения ответственности, когда в html-файле пишем только разметку, в css-файле – только стили, в js-файле – только функционал.

Рассмотрим, как при помощи js подсвечивается активная кнопка.

Последовательность действий следующая:

1. В файл style.css добавить класс active для активной кнопки и указать его стили.

Дальше работаем в файле index.js. Нам нужно:

1. Найти все кнопки используя метод querySelectorAll().
2. Убрать у всех кнопок класс active.
3. Кнопке, на которой произошёл клик - event.target - добавить класс active.

Перевод страницы на два языка

Перевод приложения, или интернационализация, – задача, часто встречающаяся в разработке. Для длинного слова internationalization часто используют сокращение i18n – i, затем ещё 18 букв, затем n.

Последовательность действий следующая.

В файле .html хранятся все элементы, внутри которых есть какой-нибудь текст, и добавляется им data-атрибут.

Параллельно с добавлением элементам атрибутов заполняется файл translate.js.

Например, в файле index.html у элемента значение data-атрибута «skill-text»:

```
<div class="skills-text" data-i18="skill-text">
  High-quality photos in the studio and on the nature
</div>
```

В файле с переводом для этого элемента указаны такие свойства:

```
const i18Obj = {
  'en': {
    'skill-text': 'High-quality photos in the studio and on the nature'
  },
  'ru' : {
    'skill-text': 'Высококачественные фото в студии и на природе'
  }
}
```

Пишется функция перевода getTranslate():

– параметром функции указываем язык, на который нужно перевести страницу, в нашем случае – английский или русский,

– в теле функции находим все элементы, содержащие data-атрибут [data-i18n];

– перебираем полученную коллекцию при помощи метода forEach();

– для каждого элемента указываем текстовое содержание – textContent – значение соответствующего этому элементу свойства объекта i18Obj;

– язык перевода – это аргумент, с которым вызывается функция getTranslate(). Ключ, по которому получаем значение, – dataset.i18n.

Переключение «светлой» и «темной» темы

Задача состоит в том, чтобы изменить цвет фона, шрифта основного текста и шрифта кнопок при наведении. Если в исходном макете был светлый фон и темный шрифт, следует указать темный фон и светлый шрифт, и наоборот.

Менять css-стили при помощи js в принципе можно, но это – не оптимальный вариант.

Более предпочтительный способ – в css указать свойства класса .light-theme (название условное) с примерно такими свойствами:

```
.light-theme {
  background-color: #fff;
  color: #000;
}

.light-theme:hover {
  color: #000;
}
```

В js создаём массив с названиями классов всех элементов, стили которых меняются при переключении темы.

При помощи метода forEach() перебираем эти элементы и при переключении на светлую тему добавляем им класс .light-theme, при переключении на тёмную тему этот класс убираем, используя метод classList.toggle().

Также переключать светлую и тёмную тему можно изменяя значения css-переменных. Например, в css переменные указываются через :root

```
:root {
  --body-color: #000;
  --text-color: #fff;
  --hover-color: #fff;
}
```

В js меняем значения переменных:

```
document.documentElement.style.setProperty('--body-color', '#fff');
document.documentElement.style.setProperty('--text-color', '#000');
document.documentElement.style.setProperty('--hover-color', '#000');
```

Хранение данных в LocalStorage

Хорошая практика - сохранить выбранные пользователем настройки в локальном хранилище браузера - localStorage. В нашем приложении такими настройками являются язык отображения страницы и светлая или тёмная тема.

Для того, чтобы их сохранить, нужно знать, какая настройка и какая тема выбраны пользователем в данный момент. Для этого создаются две глобальные переменные – lang и theme – и присваиваются им свойства по умолчанию en и light. Так как планируется, что переменные будут меняться, для их объявления используется ключевое слово let. При смене языка отображения страницы меняется значение переменной lang: en меняется на ru, при смене темы меняется значение переменной theme: light меняется на dark.

Эти переменные нужно сохранить в local storage.

Работа с local storage состоит из двух частей.

1. Сохранение данных. Могут быть два варианта: сохранять переменную при каждом её изменении или сохранить данные только перед перезагрузкой или закрытием страницы (событие beforeunload). Например:

```
function setLocalStorage() {
  localStorage.setItem('lang', lang);
}
window.addEventListener('beforeunload', setLocalStorage)
```

2. Перед загрузкой страницы (событие load) данные нужно восстановить и отобразить:
3.

```
function getLocalStorage() {
  if(localStorage.getItem('lang')) {
    const lang = localStorage.getItem('lang');
    getTranslate(lang);
  }
}
window.addEventListener('load', getLocalStorage)
```

Пояснения к коду:

- window - объект окна браузера, с ним связана загрузка и перезагрузка страницы;

– addEventListener - метод, который отлавливает событие элемента и выполняет переданную функцию;

– localStorage.setItem - метод, сохраняющий данные в localStorage. Два параметра метода: имя значения, которое сохраняется, и само значение, которое сохраняется;

– localStorage.getItem - метод получающий данные из localStorage. Параметр метода - имя, под которым сохраняется значение.

Чтобы увидеть сохранённые в браузере данные localStorage, на странице приложения следует нажать клавишу F12, на панели devTools вверху выбрать пункт Application, на боковой панели – пункт Local Storage и ссылку на страницу приложения.

Обратите внимание: данные в localStorage сохраняются в текстовом формате, например если попытаться сохранить в localStorage булевы значения true, false, вместо них localStorage вернёт строки «true» и «false».

Смена изображений

Рассмотрим как можно изменить изображение при клике по кнопке. Допустим, что в файле index.html есть кнопка и изображение:

```
<button class="btn">Winter</button>

<img src="./assets/img/autumn/1.jpg" alt="portfolio-image" class="portfolio-image">
```

Для того чтобы изменить изображение кликом по кнопке, нужно:

– найти кнопку:

```
const Btn = document.querySelector('.btn');
```

– найти изображение:

```
const portfolioImage = document.querySelector('.portfolio-image');
```

– при клике по кнопке изменить src изображения, заменив, например, autumn на winter:

```
Btn.addEventListener('click', () => {
  portfolioImage.src = "./assets/img/winter/1.jpg"
});
```

Если на странице несколько картинок, на которых нужно изменить изображение, то сделать это можно следующим образом:

– находим на странице кнопку с классом btn и записываем результат поиска в переменную portfolioBtn

```
const portfolioBtn = document.querySelector('.portfolio-btn');
```

– находим на странице все изображения с классом portfolio-image и записываем результат поиска в переменную portfolioImages:

```
const portfolioImages = document.querySelectorAll('.portfolio-image');
```

– изменяем src у всех картинок сразу:

–

```
portfolioImages.forEach((img, index) =>
        img.src = `./assets/img/winter/${index + 1}.jpg`);
```

Добавление звука на веб-страницу

С появлением HTML5 добавление аудиоплеера на страницу превратилось в рядовую задачу. Для этого достаточно в html-файл добавить код плеера, и в атрибуте src указать ссылку на аудио-файл.

```
<audio src="" controls></audio>
```

Насмотря на простоту добавления, у встроенного в браузер аудиоплеера есть недостатки. Внешний вид встроенного в браузер аудиоплеера в разных браузерах выглядит немного по-разному и очень плохо стилизуется.

Если нужно создать плеер с определённым дизайном, в приведённом выше коде необходимо убрать атрибут controls, который отвечает за отображение плеера и написать js-код для его проигрывания:

```
const audio = document.querySelector('audio');
function playAudio() {
  audio.currentTime = 0;
  audio.play();
}
function pauseAudio() {
  audio.pause();
}
```

Функции playAudio() и pauseAudio() достаточно простые. Первая запускает проигрывание плеера используя метод play(), вторая – останавливает проигрывание, используя метод pause(). Строка audio.currentTime = 0 указывает, что аудиотрек при каждом запуске функции playAudio() будет проигрываться с начала.

Есть возможность создать плеер средствами JavaScript:

```
const audio = new Audio();

function playAudio() {
```

```
  audio.src = // ссылка на аудио-файл;
  audio.currentTime = 0;
  audio.play();
}
```

У большинства плееров используется одна кнопка для проигрывания и остановки воспроизведения. Если звука нет, клик по ней запускает проигрывание звука, если звук есть - останавливает. Это юзер-френдли - пользователю так удобнее.

Чтобы реализовать такую возможность, необходимо объединить функции playAudio() и pauseAudio() в одну. Для создания такой функции прежде всего необходимо выяснить, играет ли в данный момент музыка. Для этого в глобальной области видимости, там, где находим элементы DOM, создадим переменную isPlay.

Такие переменные, названия которых начинаются с is, называются флагами. Флаг может принимать только два значения: true или false. С помощью флагов проверяем наличие или отсутствие чего-либо. В данном случае проверяем проигрывание в данный момент звука.

Когда страница только открывается, звука нет. Поэтому переменную isPlay создаём с значением false:

```
let isPlay = false;
```

При клике по кнопке Play audio необходимо не только запустить проигрывание звука, но и изменить значение переменной, указав, что теперь isPlay = true.

При клике по кнопке Stop audio останавливаем проигрывание звука и указываем, что isPlay = false.

Зная, когда проигрывается звук, а когда нет, можно создать видоизменённую функцию playAudio() которая будет по-разному работать в зависимости от того, проигрывается ли в данный момент звук. Если звука нет if(!isPlay), запускается проигрывание, иначе - останавливается.

Когда есть одна функция, которая умеет и проигрывать, и останавливать звук, в плеере достаточно оставить только одну кнопку для запуска данной функции. Наша задача научиться менять внешний вид кнопки в зависимости от того, проигрывается в данный момент звук или нет.

Возможность изменения стиля элемента средствами JavaScript применяется, только если нет других вариантов изменить стиль, например, если его значение получается динамически в результате выполнения js-кода. Для всех остальных случаев используется JavaScript для добавления или удаления у элемента класса, для которого в css-коде прописан нужный стиль.

В css-коде создаётся класс pause для которого фоновым изображением указывается иконка остановки звука. Для того чтобы на кнопке проигрывания и остановки звука иконка проигрывания менялась на иконку остановки, необходимо добавлять кнопке класс pause, когда звук

воспроизводится, и убирать, когда звука нет. Для этого используется метод classList.

Метод classList предназначен для работы с классами и поддерживается всеми современными браузерами:

– element.classList.add('class'); - добавляет элементу класс;

– element.classList.remove('class'); - удаляет класс;

– element.classList.toggle('class'); - переключает класс: добавляет, если класса нет, и удаляет, если он есть.

– element.classList.contains('class'); - проверяет, есть ли данный класс у элемента (возвращает true или false)

Так как метод classList работает только с классами, то в кавычках внутри может находиться только название класса. И вот перед этим названием класса **точка не ставится никогда**.

Для изменения стиля кнопки при клике удобно использовать метод classList.toggle().

```
function toggleBtn() {
  button.classList.toggle('pause');
}
button.addEventListener('click', toggleBtn);
```

Дополнительный материал

1 DOM-дерево https://learn.javascript.ru/dom-nodes

2 Основы JavaScript: управление DOM элементами (часть 1) https://medium.com/nuances-of-programming/основы-javascript-управление-dom-элементами-часть-1-1f13855c8b21

3 Document Object Model https://karmazzin.gitbook.io/eloquentjavascript_ru/chapter13

4 .dataset. Простейший способ хранить данные в HTML и читать их из JavaScript. https://doka.guide/js/element-dataset/

5 Использование data-* атрибутов https://developer.mozilla.org/ru/docs/Learn/HTML/Howto/Use_data_attributes

6 LocalStorage, sessionStorage https://learn.javascript.ru/localstorage

7 Web Storage API: примеры использования https://habr.com/ru/articles/496348/