Цель урока:

Вы научитесь работать с html-элементами через JavaScript-код. Также в уроке будет показано, как обрабатывать различные браузерные события. Например, чтобы при нажатии на кнопку появлялось сообщение.

Содержание урока:

1. DOM-дерево и получение HTML-элементов через JavaScript.
2. Свойства HTML-элементов.
3. Создание HTML-элементов и добавление их в DOM.
4. Остальные методы HTML-элементов (удаление элементов, работа с классами и атрибутами, метод closest).
5. Обработчики событий. Событие click.
6. Обработчики событий. Событие submit.
7. Обработчики событий. События keydown и keyup.
8. Обработчики событий. События mouseover, mouseout, mousemove.
9. Обработчики событий. События contextmenu, change, input.
10. Обработчики событий. Всплытие и погружение. Прекращение всплытия.
11. Обработчики событий. Делегирование событий.

Дополнительные материалы:

1. Все методы и свойства HTML-элемента: https://developer.mozilla.org/...
2. Браузерные события: https://learn.javascript.ru/in...
3. Продвинутая работа с селекторами: https://learn.javascript.ru/cs...

Задание #1

Вам дан HTML код.

Необходимо создать такую же структуру HTML-элемента через JavaScript и добавить итоговый элементы внутрь тега body.

Решите задачу 2-мя способами:

1. через innerHTML;
2. через document.createElement.

```
<form class="create-user-form">
    <label>
        Имя
```

```
        <input type="text" name="userName" placeholder="Введите ваше
имя">
    </label>
    <label>
        Пароль
        <input type="password" name="password" placeholder="Придумайте
Пароль">
    </label>
    <button type="submit">
        Подтвердить
    </button>
</form>
```

Задание #2

Вы вместе с командой разработки создаете приложение для контроля задач. Сейчас команда сделала только макет, чтобы приложение было красивым. Вам поручили задачу отрисовать все данные о задачах при помощи JavaScript кода (способ выберите самостоятельно). У вас имеется массив tasks и HTML-шаблон, по которому вам необходимо создать элементы для каждой задачи.

Добавьте все HTML-элементы с задачами в тег с классом **“tasks-list”.**

HTML-шаблон для задачи:

```
<div class="task-item" data-task-id="1">
   <div class="task-item__main-container">
       <div class="task-item__main-content">
           <form class="checkbox-form">
               <input class="checkbox-form__checkbox" type="checkbox"
id="task-1">
               <label for="task-1"></label>
           </form>
           <span class="task-item__text">Посмотреть новый урок по
JavaScript</span>
       </div>
       <button class="task-item__delete-button default-button
        delete-button" data-delete-task-id="5">Удалить</button>
   </div>
</div>
```

Массив задач:

```
const tasks = [
    {
        id: '1138465078061',
        completed: false,
        text: 'Посмотреть новый урок по JavaScript',
    },
    {
```

```
        id: '1138465078062',
        completed: false,
        text: 'Выполнить тест после урока',
    },
    {
        id: '1138465078063',
        completed: false,
        text: 'Выполнить ДЗ после урока',
    },
]
```

Данные об id должны использоваться в **data-task-id**, в **id** у **input** и в **for** у **label**. Значение свойства **text** должно добавляться в тег **span** с классом “**task-item__text**”.

Примечание: атрибут **for** в JavaScript коде пишется как **htmlFor.**

У вас имеются начальные файлы. Используйте их для разработки логики приложения.

Задание #3

Вы отлично справились с поставленной задачей и теперь все элементы отрисовываются правильно. Теперь вам хотят предоставить дело посложнее. Вам необходимо реализовать логику создания новых задач при помощи формы с классом “create-task-block”.

Используйте предыдущий код и повесьте обработчик событий submit на форму с классом “create-task-block”. При отправке формы создавайте новую задачу в массиве tasks и в DOM-дереве (внутри тега с классом “tasks-list”). id должен быть для каждой задачи уникальным. Текст для задачи берется из текстового поля с классом “create-task-block__input”.

Подсказка: для получения уникального id можно воспользоваться Date.now()

У вас имеются начальные файлы. Используйте их для разработки логики приложения.

Задание #4

Вы молодец! Вы справились с задачей и отдали ее на код-ревью старшему разработчику. Но старший разработчик обнаружил, что не нужно отправлять форму, если значение пустое либо задача с таким же названием уже существует. Поэтому вас просят добавить валидацию в ваш код.

Для блока с ошибкой **создавайте тег span** с классом **“error-message-block”**. Внутрь данного тега будет помещаться текст с ошибкой.

Если форма была отправлена с пустым полем, то отобразите ошибку “Название задачи не должно быть пустым”, добавив блок с ошибкой в форму с классом **“create-task-block”**. Если же задача с введенным в поле названием уже существует, то отображайте ошибку “Задача с таким названием уже существует.”

Если при отправке формы ошибок не было найдено, то удалите блок с ошибкой, если он существует в DOM, и создайте новую задачу в списке.

У вас имеются начальные файлы. Используйте их для разработки логики приложения.

Задание #5

Чтобы успешно завершить все ваши задачи на проекте, осталось выполнить только удаление задач. Для этого имеется кнопка “Удалить”. Но иногда пользователи могут случайно нажать на кнопку и из-за этого удалится задача. Для этого обычно создаются модальные окна, которые спрашивают пользователя, действительно ли он хочет удалить задачу. Сейчас вам необходимо будет реализовать такое модальное окно.

Стили для модального окна и HTML-шаблон у вас уже имеются. **Создайте HTML-элемент модального окна** с помощью JavaScript и добавьте его внутри тега **body**.

Шаблон модального окна:

```
<div class="modal-overlay modal-overlay_hidden">
   <div class="delete-modal">
       <h3 class="delete-modal__question">
           Вы действительно хотите удалить эту задачу?
       </h3>
       <div class="delete-modal__buttons">
           <button class="delete-modal__button
delete-modal__cancel-button">
               Отмена
           </button>
           <button class="delete-modal__button
delete-modal__confirm-button">
               Удалить
           </button>
       </div>
   </div>
</div>
```

Класс “**modal-overlay_hidden**” отвечает за открытие и закрытие модального окна. Если оно открыто, то класса “**modal-overlay_hidden**” нету, если закрыто, то класс “**modal-overlay_hidden**” присутствует.

Вам необходимо открывать модальное окно после нажатия на любую кнопку “Удалить” около каждой задачи. Повесьте обработчик события “**click**” на элемент с классом “**tasks-list**”. Используйте делегирование событий для того, чтобы отлавливать клики на кнопки для удаления задач.

В модальном окне есть 2 кнопки: “Отмена” и “Удалить”. “Отмена” закрывает модальное окно, а “Удалить” - удаляет задачу из массива tasks и из DOM-дерева и закрывает модальное окно.

Для удаления задачи используйте атрибут **data-task-id**, который содержит информацию о свойстве **id** объекта задачи из массива **tasks**. По данному **id** вы сможете найти нужную задачу и удалить ее.

У вас имеются начальные файлы. Используйте их для разработки логики приложения.

Задание #6

Вас поздравляет заказчик и команда разработки с успешной работой на проекте. Вы молодец! Хорошо себя показали.

Но тут один из разработчиков заболел и больше некому выполнить его задачу. Поэтому попросили вас подменить коллегу.

Вам необходимо реализовать смену темы с темной на светлую и наоборот. Тема должна переключаться при нажатии на кнопку “Tab”. Изначально тема светлая.

При смене темы вам необходимо изменить стили для следующих элементов:

1. элемент с тегом body. Если тема темная, то свойство background должно быть значением “#24292E”, иначе - “initial”.
2. Все элементы с классом “task-item”. Если тема темная, то свойство color должно быть “#ffffff”, иначе - “initial”.
3. Ко всем элементам с тегом button необходимо добавить значение “1px solid #ffffff”, если тема темная, иначе - “none”.

Для решения данной задачи используйте свойства **style** у html-элементов.

У вас имеются начальные файлы. Используйте их для разработки логики приложения.

Задание #7 (дополнительное)

Выпадающий список - популярный элемент на современном веб-сайте. Вы когда-нибудь задумывались как создается такой элемент? Сейчас вашей задачей будет реализация выпадающего списка на чистом JavaScript.

Стили и HTML-шаблон для выпадающего списка у вас уже есть.

Вам нужно добавить логику. Шаблон для выпадающего списка вам необходимо создать через **JavaScript**, используя **document.createElement**.

Шаблон выпадающего списка:

```
<div class="select-dropdown select-dropdown--123">
   <button class="select-dropdown__button
select-dropdown__button--123">
      <span class="select-dropdown select-dropdown--123">Выберите
элемент</span>
   </button>
  <ul class="select-dropdown__list select-dropdown__list--123">
```

```
      <li class="select-dropdown__list-item"
data-value="1">JavaScript</li>
      <li class="select-dropdown__list-item"
data-value="2">NodeJS</li>
      <li class="select-dropdown__list-item"
data-value="3">ReactJS</li>
      <li class="select-dropdown__list-item" data-value="4">HTML</li>
      <li class="select-dropdown__list-item" data-value="5">CSS</li>
  </ul>
</div>
```

Для реализации выпадающего списка создайте класс **CustomSelect**. Конструктор данного класса принимает 2 параметра:

1. **id** - уникальный идентификатор списка. В HTML шаблоне вместо id подставили как текст “123”. (Например, "select-dropdown__button--**123**"). Переданный параметр **id должен быть подставлен вместо данного текста**. (Например, если вы передадите в конструктор id как "**my-select**", то класс станет "select-dropdown__button**--my-select")**
2. **options** - массив вариантов выбора для выпадающего списка. Массив состоит из объектов с ключами:
    a. **value** - значение атрибута **data-value** у элемента списка (тега li)
    b. **text** - контент, переданный в тег элемента списка li. Его видит пользователь

В классе **CustomSelect** все методы и поля кроме **render** должны быть обязательно **приватными**.

Публичный метод **render** принимает в себя параметр **container** (он является DOM узлом, полученным через **document.querySelector**). В этот **container** нужно будет добавлять всю верстку выпадающего списка

Чтобы реализовать открытие и закрытие списка, вам необходимо повесить обработчик событий “**click**” на элемент с классом “**select-dropdown__button**”. Список открыт, когда у элемента с тегом **ul** есть класс “**active**”.

Чтобы реализовать выбор определенного элемента из выпадающего списка, также необходимо повесить обработчик событий "**click**". Сделать это можно двумя способами (выберите, что вам более удобно):

1. Повесить обработчик на каждый элемент списка с тегом **li**
2. Использовать делегирование событий

При клике нам нужно сохранить выбранный элемент в приватное поле **currentSelectedOption**. С помощью него мы в дальнейшем сможем получать выбранное значение. Чтобы получить значение для этого поля, ****необходимо у выбранного li получить значение атрибута "**data-value**". С помощью него мы можем найти нужный объект в массиве **options**, ****и уже этот объект сохранить в **currentSelectedOption**. Это поле будем обновлять каждый раз при выборе нового элемента меню

Текст выбранного значения подставляйте в элемент с классом “**select-dropdown**” (ключ **text** у объекта выбранного элемента)

Кроме этого к элементу **li**, по которому произошел клик, добавляйте класс “**selected**”, чтобы он становился активным и подсвечивался другим цветом. Класс “**selected**” может быть только у одного элемента с тегом **li**.

Чтобы получать текущее выбранное значение создайте геттер для приватного поля **currentSelectedOption**, который будет называться **selectedValue**. Геттер **selectedValue** должен возвращать **currentSelectedOption** - объект выбранного элемента из выпадающего списка (объект из массива **options**)

Для теста написанного класса используйте данный код:

```
const options = [
  { value: 1, text: 'JavaScript' },
  { value: 2, text: 'NodeJS' },
  { value: 3, text: 'ReactJS' },
  { value: 4, text: 'HTML' },
  { value: 5, text: 'CSS' }
];

const customSelect = new CustomSelect('123', options);
const mainContainer = document.querySelector('#container');
customSelect.render(mainContainer);
```