

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

Нижегородский государственный технический
университет им. Р.Е.Алексеева

Институт радиоэлектроники и информационных технологий
Кафедра прикладной математики и информатики

Отчёт № 6

«Работа с DOM. Формы»

по дисциплине

web-разработка

Выполнил:

студент группы 23-ПМ-1

Блинов А. С.

Проверил:

Курушин Е. А.

Нижний Новгород

2025 год

Оглавление

Цели и задачи.....	3
Теоретическая справка	4
Реализация	7
Вывод	9
Приложение	10

Цели и задачи

В ходе данной работы были поставлены следующая цель:

Изучить основные принципы взаимодействия с DOM-структурой посредством JavaScript.

Для из реализации нужно выполнить следующие задачи:

1. Получить текст любого элемента страницы.
2. Создать новый элемент р (абзац), чтобы он располагался ниже элемента из шага 1 и добавить текст (из шага 1) в созданный элемент.
3. Создать форму с элементами input и button и добавить их на страницу. Добавить созданным элементам стили CSS (все действия необходимо совершить посредством JavaScript).
4. Для кнопки (элемент button из шага 3) добавить обработчик события: Здесь необходимо проверить данные из поля input на соответствие некоему шаблону (например, валидация почты или пароля, наличие введенных данных в массиве и прочее). В результате проверки показать alert с соответствующим результатом (успешная или неуспешная проверка).
5. Добавить на страницу элемент, при нажатии на который форма (из шага 3) будет пропадать и появляться (если показана – пропадать, если скрыта – появляться). Необходимо проработать 2 варианта. В первом случае форма скрывается (display: none), во втором случае удаляется и создается.

Теоретическая справка

Разработка динамических веб-страниц требует навыков управления HTML-элементами на стороне пользователя. Основным механизмом для этого служит DOM (Document Object Model), позволяющий изменять контент, оформление и структуру страницы. Также необходимо понимать принципы обработки событий, управления атрибутами и работы с пользовательскими данными.

1. Что такое DOM?

DOM (Document Object Model) — это программное представление HTML-документа в виде иерархической структуры узлов, где каждый тег, атрибут или текстовый фрагмент становится отдельным объектом. JavaScript позволяет взаимодействовать с этими узлами: изменять их свойства, добавлять или удалять элементы.

Пример:

```
const title = document.getElementById("main-title");  
  
title.innerText = "Изменённый заголовок";
```

2. Что такое браузерные события и какие они бывают?

События — это сигналы, генерируемые браузером при взаимодействии пользователя со страницей. Основные типы событий:

- Действия мыши (click, mouseover, mouseout);
- Клавиатурные события (keypress, keydown, keyup);
- Работа с формами (focus, blur, submit, change);
- События загрузки (load, beforeunload);
- Другие (scroll, resize, touchstart).

3. Как добавить обработку события?

Для реакции на событие используется обработчик, который можно назначить через "addEventListener":

```
const btn = document.querySelector("#submit-btn");

btn.addEventListener("click", () => {

    console.log("Действие выполнено!");

});
```

Также можно использовать именованные функции или inline-обработчики в HTML.

4. Как программно добавить стили CSS? Добавить или удалить класс CSS?

Прямое изменение стилей через JavaScript:

```
const box = document.querySelector(".box");

box.style.backgroundColor = "blue";

box.style.padding = "10px";
```

Управление классами:

```
box.classList.add("highlight"); // добавить класс

box.classList.remove("hidden"); // удалить класс

box.classList.toggle("active"); // переключить класс
```

5. Как получить значение атрибута элемента? Как добавить атрибут для элемента?

Для работы с атрибутами применяются методы:

- `getAttribute("название")` — получить значение;

- `setAttribute("название", "значение")` — установить или изменить атрибут.

Пример:

```
const link = document.querySelector("a");
```

```
const url = link.getAttribute("href"); // получить ссылку
```

```
link.setAttribute("target", "_blank"); // открывать в новой вкладке
```

6. Как получить значение, введённое в элемент `input`? Как получить значение, выбранное в элементе `select`?

Значения полей формы доступны через свойство `"value"`:

- Для `"<input>"`:

```
const userInput = document.querySelector("#username").value;
```

- Для `"<select>"`:

```
const selectedOption = document.querySelector("#colors").value;
```

Реализация

В ходе выполнения лабораторного задания был проведён комплекс упражнений по изучению манипуляций с DOM-структурой, обработке форм, событий и динамического применения стилей средствами JavaScript. Все операции выполнялись программно, без использования статичной HTML-разметки — элементы генерировались, оформлялись и наделялись функционалом исключительно через код. Рабочие файл: six1.js.

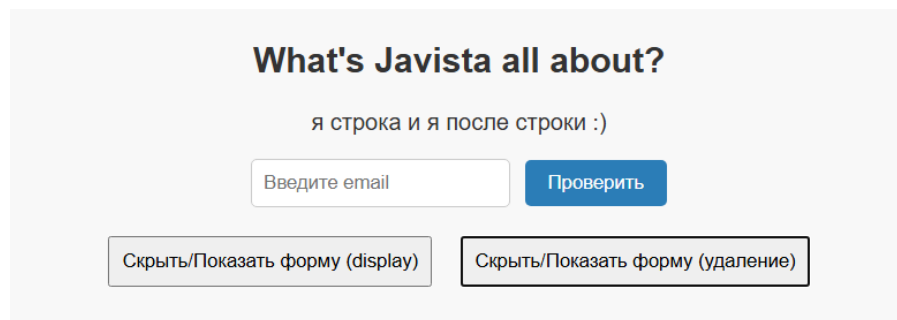
1. Извлечение текстового содержимого

На начальном этапе в документ был программно добавлен контейнер `<div>` с текстовым наполнением. Затем, используя свойство `.textContent`, текстовое содержимое было получено и записано в переменную.

2. Генерация параграфа с текстом

На основе извлечённого текста был сформирован новый элемент `<p>`, который разместили ниже исходного блока.

Визуализация: исходный текстовый блок, вывод данных в консоль и созданный параграф с скопированным содержимым отображены на рисунке 1.



What's Javista all about?

я строка и я после строки :)

Введите email

Проверить

Скрыть/Показать форму (display)

Скрыть/Показать форму (удаление)

Рисунок 1 – Вид формы

3. Динамическое создание и оформление формы

Средствами JavaScript была сгенерирована форма, включающая текстовое поле и кнопку отправки. Стилизовое оформление элементов задавалось программно через `element.style`.

4. Обработка событий и проверка ввода

Для кнопки формы был реализован обработчик события `click`. При активации происходит проверка введённых данных на соответствие формату e-mail с применением регулярного выражения. Результат валидации отображается через `alert`.

5. Управление видимостью формы

Вариант 1: Переключение видимости

Добавлена управляющая кнопка, изменяющая свойство `display` формы для её скрытия или отображения.

Вариант 2: Полная регенерация

Реализована дополнительная кнопка, которая полностью удаляет форму из DOM и при повторном нажатии восстанавливает её заново.

Вывод

Все задачи были эффективно выполнены с применением нативного JavaScript. В процессе разработки осуществлялось взаимодействие с элементами веб-страницы, включая генерацию и управление интерактивными формами, настройку визуального оформления, обработку действий пользователя и проверку корректности вводимых данных.

Приложение

```
// === 1. Получаем заголовок секции ===
const aboutTitle = document.getElementById("about-title");

// === 2. Добавляем абзац после заголовка ===
const newParagraph = document.createElement("p");
newParagraph.textContent = "я строка и я после строки :)";
aboutTitle.insertAdjacentElement("afterend", newParagraph);

// === 3. Функция создания формы ===
function createEmailCheckForm() {
    const form = document.createElement("form");
    form.id = "email-check-form";

    const input = document.createElement("input");
    input.type = "text";
    input.placeholder = "Введите email";
    input.style.padding = "8px";
    input.style.marginRight = "10px";
    input.style.border = "1px solid #ccc";
    input.style.borderRadius = "4px";

    const button = document.createElement("button");
    button.textContent = "Проверить";
    button.type = "submit";
    button.style.padding = "8px 16px";
    button.style.backgroundColor = "#2b7db7";
    button.style.color = "#fff";
    button.style.border = "none";
    button.style.borderRadius = "4px";
    button.style.cursor = "pointer";

    form.appendChild(input);
    form.appendChild(button);

    // Валидация email при отправке
    form.addEventListener("submit", function (e) {
        e.preventDefault();
        const value = input.value.trim();
        const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

        if (emailPattern.test(value)) {
            alert("✓ Email введён корректно!");
        } else {
            alert("✗ Неверный формат email.");
        }
    });

    // Вставим форму после абзаца
    newParagraph.insertAdjacentElement("afterend", form);
    return form;
}

// === 4. Сразу создаём форму для варианта 1 ===
let form = createEmailCheckForm();
let form2 = form;
let isFormCreated = true;

// === 5. Кнопка скрытия через display ===
const toggleDisplayBtn = document.createElement("button");
toggleDisplayBtn.textContent = "Скрыть/Показать форму (display)";
toggleDisplayBtn.style.marginTop = "20px";
```

```

toggleDisplayBtn.style.marginRight = "10px";
toggleDisplayBtn.style.padding = "8px";
toggleDisplayBtn.style.cursor = "pointer";

// Вставляем кнопку после формы
form.insertAdjacentElement("afterend", toggleDisplayBtn);

toggleDisplayBtn.addEventListener("click", () => {
  form.style.display = (form.style.display === "none") ? "block" : "none";
});

// === 6. Кнопка удаления и создания формы ===
const toggleCreateBtn = document.createElement("button");
toggleCreateBtn.textContent = "Скрыть/Показать форму (удаление)";
toggleCreateBtn.style.marginTop = "20px";
toggleCreateBtn.style.marginLeft = "10px";
toggleCreateBtn.style.padding = "8px";
toggleCreateBtn.style.cursor = "pointer";

// Вставляем кнопку после предыдущей кнопки
toggleDisplayBtn.insertAdjacentElement("afterend", toggleCreateBtn);

// Сохраняем текущее состояние display формы
let currentDisplayStyle = "block";

toggleCreateBtn.addEventListener("click", () => {
  if (isFormCreated) {
    // Сохраняем текущий display перед удалением
    currentDisplayStyle = form.style.display;

    // Удаляем форму
    form2.remove();
    isFormCreated = false;
  } else {
    // Воссоздаем форму и применяем сохраненный стиль
    form2 = createEmailCheckForm();
    form2.style.display = currentDisplayStyle; // Восстановим сохраненный display
    isFormCreated = true;
  }
});

```