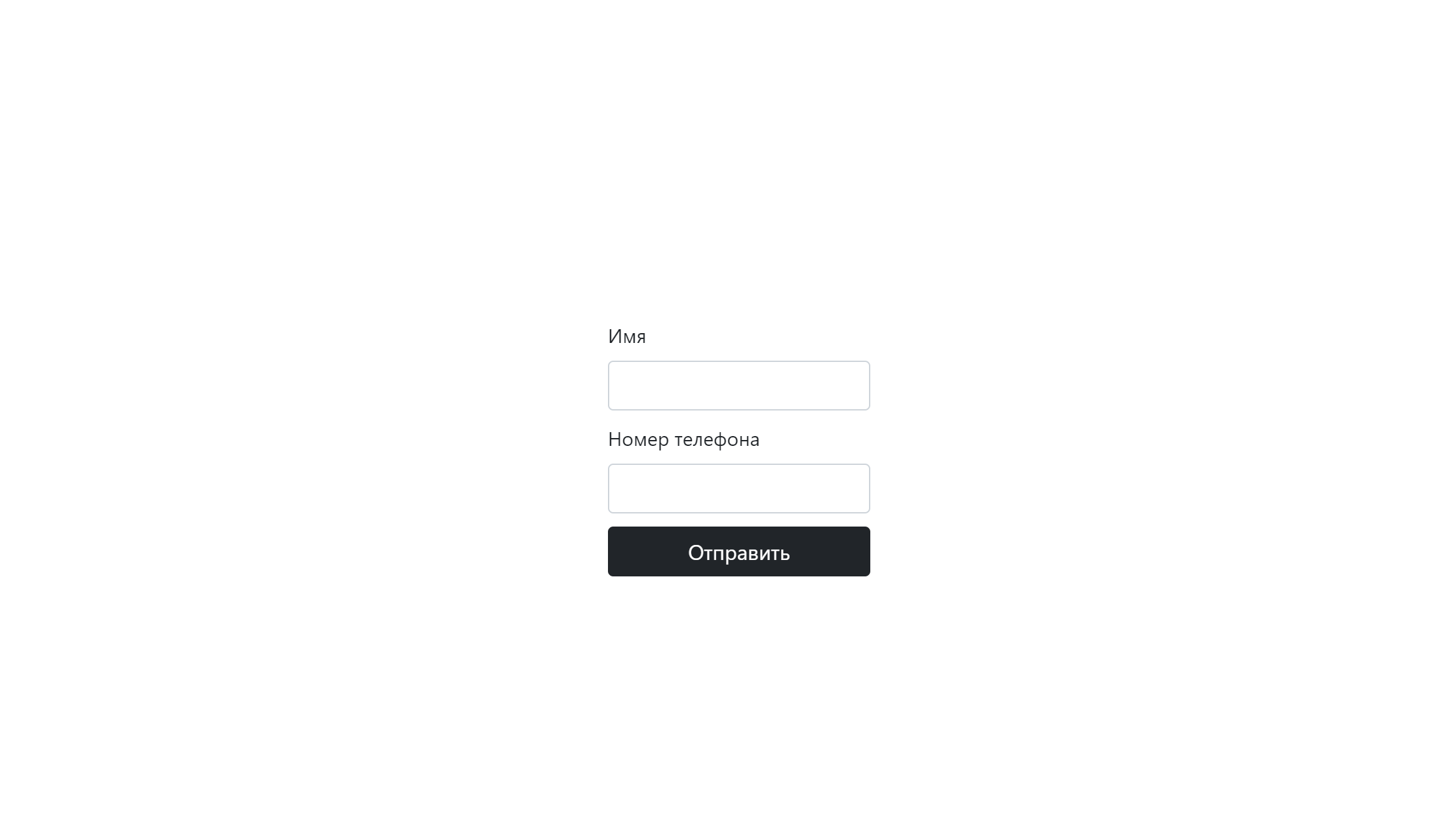
**Взаимодействие с DOM**

**Введение**

JavaScript предоставляет множество методов для работы с Document Object Model или сокращенно DOM. Они используются для доступа к элементам, находящимся на странице, для изменения состояния и содержимого страницы. В курсе мы разберем все основные приемы и функции, которые используются для работы с DOM.

Как пример для работы возьмем страницу из предыдущего урока:

<!DOCTYPE *HTML*>  
<html *lang*="ru">  
<head>  
 <title>Форма</title>  
 <link *href*="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" *rel*="stylesheet"  
 *integrity*="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" *crossorigin*="anonymous">  
</head>  
<style>  
 p {  
 font-size: 15px;  
 padding-top: 10px;  
 margin-bottom: 0;  
 }  
  
 #div\_form {  
 position: fixed;  
 width: 100vw;  
 height: 100vh;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 }  
  
 #form, .btn {  
 width: 200px;  
 }  
  
</style>  
<body>  
<div *id*="div\_form">  
 <form *id*="form">  
 <div *class*="form-label" >  
 <p *id*="name">Имя</p>  
 </div>  
 <label>  
 <input *class*="form-control">  
 </label>  
 <div *class*="form-label">  
 <p *id*="tel">Номер телефона</p>  
 </div>  
 <label>  
 <input *class*="form-control">  
 </label>  
 <p></p>  
 <label>  
 <button *class*="btn btn-dark">  
 Отправить  
 </button>  
 </label>  
 </form>  
</div>  
</body>  
</html>



**Document**

Начнем с основ:

Для работы со страницей используется глобальный объект - document. Основные его методы:

* getElementById(“id”) – возвращает элемент с id = “id”, то есть тег, атрибуты и содержимое этого объекта.

Например:  
*const log* = console.log;  
*log*(document.getElementById("tel")); *// <p id="tel">Номер телефона</p>*

* getElementsByClassName(“class”) – возвращает набор из элементов с class = “class”.

Например:  
*const log* = console.log;  
*const* forms = document.getElementsByClassName("form-control")  
  
*for* (*let* element *of* forms) {  
 *log*(element);  
 *//<input class="form-control">  
 //<input class="form-control">*}

* getElementsByTagName(“tag”) – возвращает набор из элементов <tag></tag>.

Например:  
*const log* = console.log;  
*const* forms = document.getElementsByTagName("p")  
  
*for* (*let* element *of* forms) {  
 *log*(element);  
 *//<p id="name">Имя</p>  
 //<p id="tel">Номер телефона</p>  
 //<p></p>*}

* createElement(“tag”) – cоздает элемент <tag></tag>.

Например:  
*const* div = document.createElement("div");

Как и любой объект в JavaScript его можно записать в переменную. Чтобы ваш код был понятен, воспользуйтесь переменной “$”:

const $ = document

Важно понимать, что его методы возвращают значения в виде ссылки на объект, то есть вместе с изменением полученного объекта будет меняться и его состояние на странице. Такая ссылка называется Node, а их набор NodeList. Внимание, методы Array не распространяются на NodeList.

Также первые три метода можно использовать с любым Node. Делается это для сужения круга поиска и ускорения программы. Поиск будет воспроизводиться внутри Node и не выйдет за его пределы.

**Node**

**Методы и свойства Node**

Для примера будем использовать третий <p></p> на нашей странице. Получить его можно с помощью const p = document.getElementsByTagName(“p”)[2]

Свойства:

* p.id – свойство, позволяющее получить или изменить id элемента.
* p.textContent – свойство, позволяющее получить или изменить текст элемента.
* p.firstChild – Возвращает первый дочерний Node или null, если узел их не имеет.
* p.lastChild – Возвращает последний дочерний Node или null, если узел их не имеет.
* p.parentNode – Возвращает “родителя” (Node) или null, если узел их не имеет.
* p.style – свойство, отвечающее за атрибут style внутри элемента. В данный момент его использование считается моветоном. Но в редких случаях без него не обойтись. Это свойство имеет столько разветвлений, сколько есть СSS свойств. Проще разобраться на примере:

Допустим, имеется элемент:

<p id = “my\_p” style =”font-size: 20px;color: black;”>Абзац</p>

Чтобы поменять цвет, нам нужно изменить свойство style.color объекта p, например:

p.style.color="red";

И теперь наш элемент выглядит таким образом:

<p id = “my\_p” style =”font-size: 20px;color: red;”>Абзац</p>

Методы:

1. p.ClassList – семейство методов для доступа и изменения классов объекта.
   1. add(“class”) – добавить класс “class” объекту.
   2. remove(“class”) – удалить класс “class” у объекта.
   3. contains(“class”) – проверить наличие класса “class” у объекта (возвращает true/false).
   4. toggle(“class”) – работает согласно своему названию, как переключатель. Если отсутствует – добавим, в противном случае удалим. Заменяет конструкцию if(!contains) -> add, else -> remove.
2. p.cloneNode() – используется для клонирования p, так как при обычном приравнивании вы получите тот же Node.
3. p.appendChild(node) – добавляет node в список дочерних элементов p.
4. p.contains(node ) – проверяет, является ли node дочерним элементом p. (возвращает true/false)
5. p.hasChildNodes() – проверяет, имеет ли p дочерние элементы. (возвращает true/false)
6. p.removeChild(node) – удаляет node из списка дочерних элементов p. Работает в связке с hasChildNodes
7. p.replaceChild(new\_сhild, old\_сhild) – заменяет дочерний элемент old\_сhild на выбранный new\_сhild. Возвращает заменённый элемент.

**Обработчики событий и взаимодействие с пользователем**

Слушатели или обработчики (addEventListener):

addEventListener() – это еще один метод Node. Но его нужно рассмотреть отдельно, так как он имеет несколько ответвлений. Классический cинтаксис:

p.addEventListener(type, listener);

где

p – Node, к которому добавляют слушатель.

type – тип события, при котором должен сработать listener.

listener – callBack функция , которая и будет производить изменения на странице при срабатывании слушателя.

Рассмотрим основные события мыши (type):

* “click” – одинарное нажатие левой кнопкой мыши.
* “contextmenu” – одинарное нажатие правой кнопкой мыши.
* “dblclick” – двойное нажатие левой кнопкой мыши.
* “mousedown” – на элементе нажимается кнопка указывающего устройства.
* “mouseenter” – указывающее устройство перемещено на элемент, к которому подключён обработчик.
* “mouseleave” – указывающее устройство перемещается от элемента, к которому подключён обработчик.
* “mousemove” – указывающее устройство перемещается по элементу. (Происходит непрерывно при движении мыши.)
* “mouseenter” – указывающее устройство перемещено на элемент, к которому подключён обработчик.

Рассмотрим основные способы вывода и ввода (для пользователя) информации в браузере:

1. alert(“message”) – классическое сообщение, всплывающее окно.
2. prompt(title, default) – запрос на ввод, всплывающее окно с заголовком title, полем для ввода и значением по умолчанию default.
3. confirm(question) - окно с вопросом question с двумя кнопками: OK и CANCEL.
4. <input> - давайте поподробнее разберем работу с полем ввода.
   1. Атрибут type – отвечает за встроенные в HTML проверки и стили. Например, если вы хотите, чтобы вводимое слово превращалось в набор точек, как при вводе пароля, то используйте <input type = “password”>.

Изображение выглядит как текст

Автоматически созданное описание

Пример для электронной почты: <input type = “email”>.

Изображение выглядит как текст

Автоматически созданное описание

* 1. Атрибут placeholder – отвечает за значение-подсказку, которое будет показано серым цветом и исчезнет при начале ввода. Например: <input type = “password” placeholder= ”Придумайте пароль”>.

Изображение выглядит как текст

Автоматически созданное описание

* 1. Атрибут maxlength/ minlength – отвечает за максимальное/минимальное количество введенных символов. Например: <input type = “password” placeholder=”Придумайте пароль” minlength=”10”>.

Изображение выглядит как текст

Автоматически созданное описание

* 1. Получение информации из <input> средствами языка JavaScript: получив Node любым из представленных выше способов, используйте свойство value для получения или изменения значения. Например const inputValue = $.getElementById(“input”).value;

**Примеры использования**

Давайте попробуем воссоздать нашу страницу, оставив только теги <body>, <head>, <style>. Все остальное добавим с помощью JavaScript. Сделать это можно двумя способами: использовать тег <script> в файле html или подключить внешний файл index.js. Принято использовать второй способ, но для наглядности мы воспользуемся первым.

Код шаблона для тренировки:

<!DOCTYPE *HTML*>  
<html *lang*="ru">  
<head>  
 <title>Форма</title>  
 <link *href*="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" *rel*="stylesheet"  
 *integrity*="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" *crossorigin*="anonymous">  
</head>  
<style>  
 p {  
 font-size: 15px;  
 padding-top: 10px;  
 margin-bottom: 0;  
 }  
  
 #div\_form {  
 position: fixed;  
 width: 100vw;  
 height: 100vh;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 }  
  
 #form, .btn {  
 width: 200px;  
 }  
  
</style>  
<body>

<script>

</script>  
</body>  
</html>

1. Сначала создадим все элементы с помощью document.createElement():

*const* $ = document;  
 *const* divForm = $.createElement("div");

*const* form = $.createElement("form");

//блок для хранения полей

*const* inputList = $.createElement("div")

//блок для имени  
 *const* nameDivForP = $.createElement("div");  
 *const* nameP = $.createElement("p");  
 *const* nameLabel = $.createElement("label")  
 *const* nameInput = $.createElement("input")

//блок для телефона  
 *const* telDivForP = $.createElement("div");  
 *const* telP = $.createElement("p");  
 *const* telLabel = $.createElement("label")  
 *const* telInput = $.createElement("input")  
   
 *const* thirdP = $.createElement("p");

//блок для кнопки  
 *const* buttonLabel = $.createElement("label")  
 *const* button = $.createElement("button")

1. Теперь из созданных элементов “построим” древо и добавим его дочерним в body:

*//Создание древа элементов и загрузка его в body*divForm.appendChild(form);  
 inputList.appendChild(nameDivForP);  
 nameDivForP.appendChild(nameP);  
 inputList.appendChild(nameLabel);  
 nameLabel.appendChild(nameInput);  
 inputList.appendChild(telDivForP);  
 telDivForP.appendChild(telP);  
 inputList.appendChild(telLabel);  
 telLabel.appendChild(telInput);

form.appendChild(inputList);  
 form.appendChild(thirdP);

form.appendChild(buttonLabel);  
 buttonLabel.appendChild(button);  
*//обратимся к body с помощью встроенного свойства document.body*$.body.appendChild(divForm);

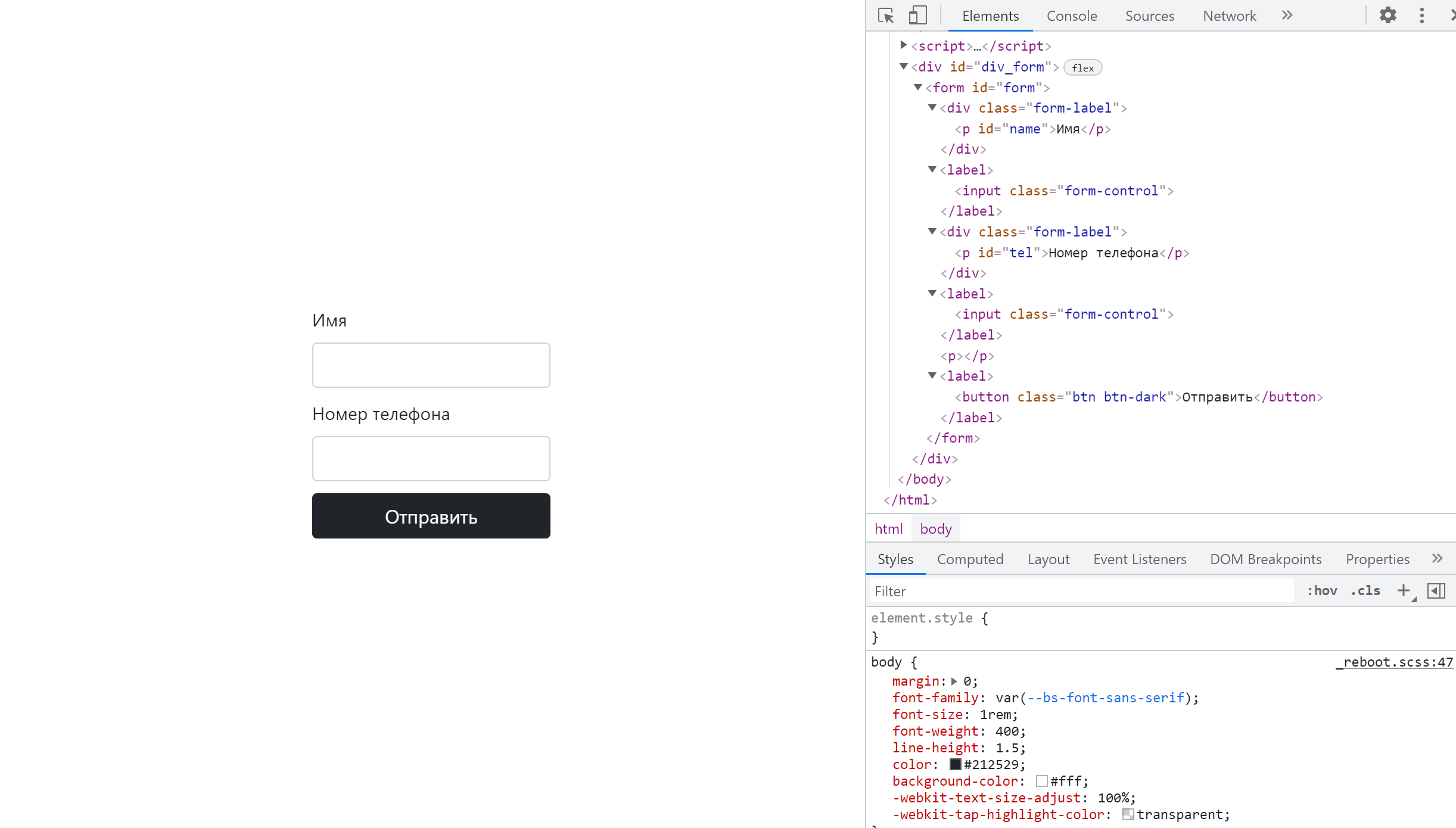
В коде специально добавлена лишняя табуляцию, чтобы иерархия была более наглядной.

1. Последним шагом добавим текст, все классы и идентификаторы элементов.

*//Текст на странице*nameP.textContent = "Имя";  
telP.textContent = "Номер телефона";  
button.textContent = "Отправить";  
  
*//Идентификаторы*divForm.id = "div\_form"  
form.id = "form";  
nameP.id = "name";  
telP.id = "tel";

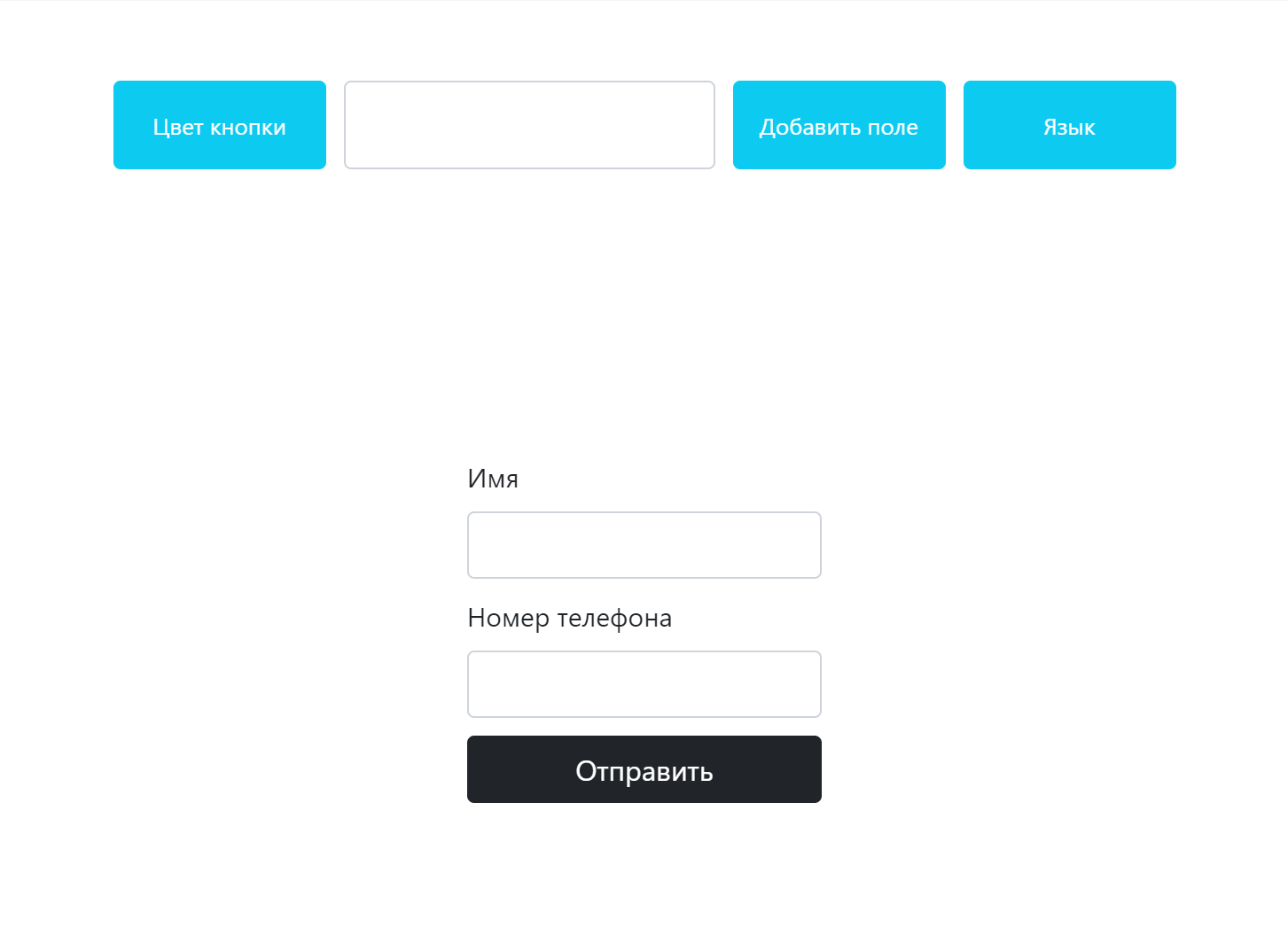
inputList.id = "input\_list";  
  
*//Классы*nameDivForP.classList.add("form-label");  
telDivForP.classList.add("form-label");  
nameInput.classList.add("form-control");  
telInput.classList.add("form-control");  
button.classList.add("btn");  
button.classList.add("btn-dark");

Результат работы нашего скрипта создает первоначальную страницу:



Для отработки оставшегося материала добавим к нашей странице дополнительную панель. На ней будет находиться 3 кнопки и одно поле для изменения стиля формы:

* Для изменения цвета кнопки.
* Для добавления новых полей.
* Для перевода страницы на Английский.



1. В первую очередь подготовим стили и присвоим кнопе “Отправить” id = “submit\_btn” :  
   */\*Задаем меню размер, позицию и адаптивность\*/*#menu\_change {  
    position: fixed;  
    width: 100vw;  
    height: 20vh;  
    display: flex;  
    justify-content: center;  
   }  
   */\*Разделим меню на 3 блока по 350px и сделаем их так же адаптивными\*/*.block {  
    display: flex;  
    align-items: center;  
    max-width: 350px;  
    flex-wrap: nowrap; */\*Чтобы кнопка не уходила под поле ввода\*/*}  
   */\*Подготовим кнопки\*/*.block .btn {  
    max-width: 120px;  
    font-size: 13px;  
    min-height: 50px;  
    margin: 0 5px;  
    color: white;  
   }  
   */\*Подготовим поле ввода\*/*.block input {  
    min-height: 50px;  
    margin: 0 5px  
   }
2. Реализуем меню средствами языка JavaScript:

*const* $ = document;  
*//Создание элементов  
const* menu = $.createElement("div");  
*const* firstBlock = $.createElement("div");  
*const* secondBlock = $.createElement("div");  
*const* thirdBlock = $.createElement("div");  
*const* btnColorChange = $.createElement("button");  
*const* btnNewInput = $.createElement("button");  
*const* newInput = $.createElement("input");  
*const* btnLangChange = $.createElement("button");  
*//Инициализация древа*menu.appendChild(firstBlock);  
firstBlock.appendChild(btnColorChange);  
menu.appendChild(secondBlock);  
secondBlock.appendChild(newInput);  
secondBlock.appendChild(btnNewInput);  
menu.appendChild(thirdBlock);  
thirdBlock.appendChild(btnLangChange);  
*//Текст*[btnColorChange.textContent, btnNewInput.textContent, btnLangChange.textContent] = ["Цвет кнопки", "Добавить поле", "Язык"];  
*//Идентификаторы*menu.id = "menu\_change";  
btnColorChange.id = "btn\_color-change\_id";  
btnNewInput.id = "btn\_new-input\_id";  
newInput.id = "input\_new-input\_id";  
btnLangChange.id = "btn\_lang-change\_id";  
*//Классы  
for* (*let* element *of* [btnColorChange, btnNewInput, btnLangChange]) {  
 element.classList.add("btn");  
 element.classList.add("btn-info");  
}  
newInput.classList.add("form-control");  
*for* (*let* element *of* [firstBlock, secondBlock, thirdBlock]) {  
 element.classList.add("block");  
}  
$.body.appendChild(menu);

1. Добавим слушатели по клику с соответствующими проверками:

*//Слушатели*btnColorChange.addEventListener("click", () => {  
 *const* btn = $.getElementById("submit\_btn");  
 *//Поменяем черный на красный или наоборот* btn.classList.toggle("btn-dark");  
 btn.classList.toggle("btn-danger");  
})  
btnNewInput.addEventListener("click", () => {  
 *const* nameLabel = $.getElementById("input\_new-input\_id").value; *//Веденое юзером название  
//Прекратим, если поле пустое или только из пробелов  
 if* (nameLabel.trim() === "") {  
 *alert*("Поле пустое");  
 *return*;  
 }  
 *const* inputList\_ = $.getElementById("input\_list"); *//Набор полей  
//Прекратим, если такое поле уже есть  
 for* (*let* element *of* inputList\_.getElementsByTagName("p")) {  
 *if* (element.textContent.toUpperCase() === nameLabel.toUpperCase()) {  
 *alert*("Такое поле уже есть");  
 *return*;  
 }  
 }  
  
 *const* newLabel = $.createElement("div");  
 *const* newLabelP = $.createElement("p");  
 *const* newInputLabel = $.createElement("label");  
 *const* newInput = $.createElement("input");  
  
 newLabel.appendChild(newLabelP);  
 newInputLabel.appendChild(newInput);  
  
 newLabel.classList.add("form-label");  
 newInput.classList.add("form-control");  
 newLabelP.textContent = $.getElementById("input\_new-input\_id").value;  
 newInput.id = inputList\_.childElementCount + "\_users\_p"  
  
 inputList\_.appendChild(newLabel);  
 inputList\_.appendChild(newInputLabel);  
})  
  
btnLangChange.addEventListener("click", () => {  
 *const* btn = $.getElementById("submit\_btn");  
 *const* name = $.getElementById("name");  
 *const* tel = $.getElementById("tel");  
 *if* (btnColorChange.textContent !== "Цвет кнопки") {  
 [btnColorChange.textContent, btnNewInput.textContent, btnLangChange.textContent] = ["Цвет кнопки", "Добавить поле", "Язык"];  
 [name.textContent, tel.textContent, btn.textContent] = ["Имя", "Номер телефона", "Отправить"];  
 } *else* {  
 [btnColorChange.textContent, btnNewInput.textContent, btnLangChange.textContent] = ["Button color", "New Input", "Language"];  
 [name.textContent, tel.textContent, btn.textContent] = ["Name", "Phone number", "Submit"];  
 }  
})

Работы приложения:

