Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Отчёт
по лабораторной работе №2
«Работа с *JavaScript*»
по дисциплине «Технологии интернет-программирования»

Выполнил:
студент гр. 820601
Шведов А. Р.

Проверил:
Н. В. Хаджинова

Минск 2021

СОДЕРЖАНИЕ

Введение 3
1 Цель работы 4
2 Теоретическая часть 5
2.1 Объектная модель браузера 5
2.2 Методы *alert*(), *confirm*() и *prompt*() объекта *window* 6
2.3 Основы языка *JavaScript* 7
3 Практическая часть 13
3.1 Обновление структуры проекта 13
3.2 Руководство пользователя 14
Заключение 20
Список использованных источников 21

ВВЕДЕНИЕ

Для придания веб-страницам динамизма (например, выпадающие меню, анимации) используются языки написания скриптов. Стандартным скриптовым языком во всемирной паутине является *JavaScript*. Он обычно используется как встраиваемый язык для программного доступа к объектам приложений, а наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

JavaScript – это легковесный, интерпретируемый или *JIT*-компилируемый, объектно-ориентированный язык с функциями первого класса. Наиболее широкое применение находит как язык сценариев веб-страниц, но также используется и во множестве других программных продуктов. Это прототипно-ориентированный, мультипарадигменный язык с динамической типизацией, который поддерживает объектно-ориентированный, императивный и декларативный стили программирования.

1 ЦЕЛЬ РАБОТЫ

Ознакомиться с основами языка программирования *JavaScript* и возможностями его применения на веб-страницах. Создать новые страницы сайта, на которых продемонстрировать работу методов *alert*(), *confirm*() и *prompt*(), а также работу с переменными примитивных типов, объектами, массивами и функциями.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 Объектная модель браузера

Веб-страницы бывают статическими и динамическими, последние отличаются тем, что в них используются сценарии (программы) на языке *JavaScript*.

Для того чтобы изменять уже нарисованный браузером экран или создавать новые окна, нужно проникнуть в иерархию объектов браузера, получить доступ к его объектам. В сценариях *JavaScript* браузер предоставляет веб-разработчику множество «готовых» объектов, с помощью которых он может взаимодействовать с элементами веб-страницы и самим браузером. В совокупности же все эти объекты составляют объектную модель браузера (*BOM – Browser Object Model*).

Схема объектной модели браузера приведена на рисунке 1.

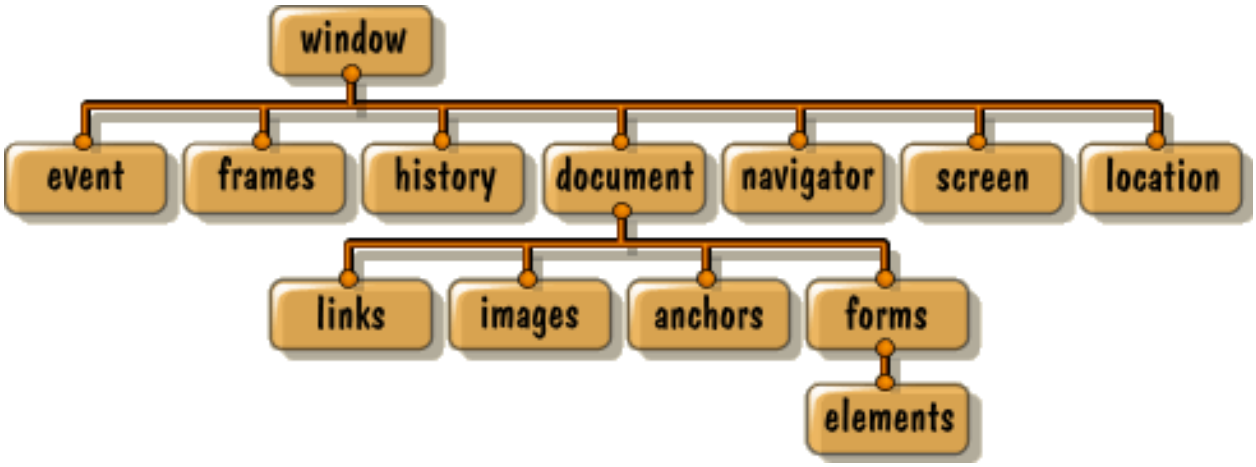

Рисунок 1 – Схема объектной модели браузера

На самом верху этой модели находится глобальный объект *window*. Он представляет собой одно из окон или вкладку браузера с его панелями инструментов, меню, строкой состояния, *HTML*-страницей и другими объектами. Доступ к этим различным объектам окна браузера осуществляется с помощью следующих основных объектов: *navigator*, *history*, *location*, *screen*, *document* и некоторых других. Так как данные объекты являются дочерними по отношению к объекту *window*, то обращение к ним происходит как к свойствам объекта *window*.

Из всех этих объектов наибольший интерес и значимость для разработчика представляет объект *document*, который является корнем объектной модели документа (*DOM – Document Object Model*). Данная модель в отличие от объектной модели браузера стандартизована в спецификации и поддерживается всеми браузерами. Объект *document* представляет собой *HTML*-документ, загруженный в окно (вкладку) браузера. С помощью свойств и методов данного объекта можно получить доступ к содержимому *HTML*-документа, а также изменить его содержимое, структуру и оформление.

2.2 Методы *alert*(), *confirm*() и *prompt*() объекта *window*

Объект *window* является главным в браузере. Он является корнем иерархии всех объектов, доступных веб-разработчику в сценариях *JavaScript*, и представляет собой окно в браузере, причём каждая вкладка содержит свой собственный объект *window*.

Объект *window* кроме глобальных объектов имеет собственные свойства и методы, которые предназначены для следующих задач:

– открытия нового окна (вкладки);

– закрытия окна (вкладки) с помощью метода *close*();

– распечатывания содержимого окна (вкладки);

– передачи фокуса окну или для его перемещения на задний план (за всеми окнами);

– управления положением и размерами окна, а также для осуществления прокручивания его содержимого;

– изменения содержимого статусной строки браузера;

– взаимодействия с пользователем посредством всплывающих окон;

– выполнения определённых действий через определённые промежутки времени и некоторых других задач.

Для взаимодействия с пользователем у объекта *window* присутствуют методы *alert*(), *confirm*() и *prompt*().

Метод *alert*() предназначен для вывода в браузере предупреждающего модального диалогового окна с некоторым сообщением и кнопкой «ОК». При его появлении дальнейшее выполнение кода страницы прекращается до тех пор, пока пользователь не закроет это окно. Кроме этого, оно также блокирует возможность взаимодействия пользователя с остальной частью страницы. Метод имеет единственный аргумент (*message*) – текст сообщения, которое необходимо вывести в модальном диалоговом окне. В качестве результата *alert*() ничего не возвращает.

Метод *confirm*() применяется для вывода модального диалогового окна с сообщением и кнопками «ОК» и «Отмена». Оно обычно используется для запроса у пользователя разрешения на выполнение того или иного действия. Если пользователь нажал на кнопку «ОК», то в качестве результата возвращается *true*, в остальных случаях – *false*.

Метод *prompt*() предназначен для вывода диалогового окна с сообщением, текстовым полем для ввода данных и кнопками «ОК» и «Отмена». Это окно предназначено для запроса данных, которые пользователю нужно ввести в текстовое поле. В качестве результата возвращается значение, введённое пользователем или *null*. Если пользователь не ввёл данные (поле ввода пустое) и нажал на «ОК», то будет возвращена пустая строка. В качестве второго параметра метода можно передать строку, содержащую значение по умолчанию, отображаемую в поле ввода текста.

Синтаксис использования методов представлен ниже:

```
alert(message);
const confirm_result = confirm(question);
const prompt_result = prompt(message, default);
```

2.3 Основы языка JavaScript

2.3.1 В *JavaScript* имеются следующие примитивные типы данных: *number, boolean, string, undefined, null*. Сразу нужно отметить, что при работе с примитивными типами данных, например, со строковыми литералами, даже не проводя явного преобразования, можно обращаться к их методам и свойствам. При попытке выполнения подобных операций литералы автоматически оснащаются соответствующей объектной обёрткой.

2.3.2 В *JavaScript* имеется лишь один тип чисел — это числа двойной точности с плавающей запятой. Поэтому результаты вычисления некоторых выражений имеют соответствующую погрешность.

В *JavaScript* имеется объект *Number*, представляющий собой объектную обёртку для числовых значений.

Существуют глобальные функции, предназначенные для преобразования значений других типов в числовой тип. Это — *parseInt*(), *parseFloat*() и конструкция *Number*(), которая в данном случае выступает в виде обычной функции, выполняющей преобразование типов.

Если в ходе операции с числами получается нечто, не являющееся числом (в ходе неких вычислений, или при попытке преобразования чего-либо в число), *JavaScript* не выдаст ошибку, а представит результат подобной операции в виде значения *NaN* (*Not-a-Number*, не число). Для того, чтобы проверить, является ли некое значение *NaN*, можно воспользоваться функцией *isNaN*().

Арифметические операции *JavaScript* работают вполне привычным образом, но надо обратить внимание на то, что оператор «+» может выполнять и сложение чисел, и конкатенацию строк.

2.3.3 Строки в JavaScript представляют собой последовательности символов *Unicode*. Строковые литералы создают, заключая текст, который нужно в них поместить, в двойные или одинарные кавычки.

Как уже было сказано, при работе со строковыми литералами можно полагаться на соответствующую объектную обёртку, в прототипе которой имеется множество полезных методов, среди которых – *substring*(), *indexOf*(), *concat*().

Строки, как и другие примитивные значения, не изменяемы (*immutable*). Например, метод *concat*() не модифицирует существующую строку, а возвращает в виде результата новую, полученную в результате конкатенации.

Для проверки соответствия строк некоторым шаблонам в языке присутствуют регулярные выражения.

2.3.4 Логический тип данных в *JavaScript* представлен двумя значениями – *true* и *false*. Язык может автоматически преобразовывать различные значения к логическому типу данных. Так, ложными, помимо логического значения false, являются значения *null, undefined*, " (пустая строка), 0 и *NaN*. Всё остальное, включая любые объекты, представляет собой истинные значения.

В ходе выполнения логических операций всё, что считается истинным, преобразуется к *true*, а всё, что считается ложным, преобразуется к *false*.

2.3.5 Объекты – это динамические структуры, состоящие из пар ключ-значение. Значения могут иметь примитивные типы данных, могут быть объектами или функциями.

Объекты проще всего создавать, используя синтаксис объектных литералов:

```
let obj = {
  message : "A message",
  doSomething : function() {}
}
```

Свойства объекта можно, в любое время, читать, добавлять, редактировать и удалять.

Объекты в языке реализованы в виде хэш-таблиц. Простую хэш-таблицу можно создать, используя команду *Object.create(null)*.

Если объект нужно сделать неизменяемым, можно воспользоваться командой *Object.freeze()*.

Для перебора всех свойств объекта можно воспользоваться командой *Object.keys()*.

2.3.6 В *JavaScript* переменные можно объявлять, используя ключевые слова *var*, *let* и *const*.

При использовании ключевого слова *var* можно объявить переменную, и, если надо, инициализировать её каким-то значением. Если переменная не инициализирована, её значением является *undefined*. Переменные, объявленные с использованием ключевого слова *var*, имеют функциональную область видимости.

Ключевое слово *let* очень похоже на *var*, разница заключается в том, что переменные, объявленные с ключевым словом *let*, имеют блочную область видимости.

Блочную область видимости имеют и переменные объявленные с помощью ключевого слова *const*, которые, учитывая то, что значения подобных переменных нельзя изменять, правильнее будет называть «константами». Ключевое слово *const*, которое «замораживает» значение переменной, объявленной с его использованием, можно сравнить с методом *Object.freeze*(), «замораживающим» объекты.

Если переменная объявлена за пределами какой-либо функции, её область видимости является глобальной.

2.3.7 Массивы в *JavaScript* реализованы с использованием объектов. Как результат, говоря о массивах, фактически, подразумевают объекты, похожие на массивы. Работать с элементами массива можно, используя их индексы. Числовые индексы преобразуются в строки и используются как имена для доступа к значениям элементов массивов. Например, конструкция вида *arr*[1] аналогична конструкции вида *arr*['1'], обе конструкции дадут

доступ к одному и тому же значению: *arr*[1] === *arr*['1']. В соответствии с вышесказанным, простой массив, объявленный командой *let arr* = ['*A*', '*B*', '*C*'], представляется в виде объекта примерно следующего вида:

```
{
  '0': 'A',
  '1': 'B',
  '2': 'C'
}
```

Удаление элементов массива с использованием команды *delete* оставляет в нём «дыры». Для того чтобы избежать этой проблемы, можно использовать команду *splice*(), но работает она медленно, так как, после удаления элемента, перемещает оставшиеся элементы массива, сдвигая их влево.

Методы массивов позволяют легко реализовывать такие структуры данных, как стеки и очереди.

2.3.8 Функции в *JavaScript* являются объектами. Функции можно назначать переменным, хранить в объектах или массивах, передавать в виде аргументов другим функциям и возвращать из других функций.

Существует три способа объявления функций:

– Классическое объявление функции (*Function Declaration или Function Statement*);

– Использование функциональных выражений (*Function Expression*), которые ещё называют функциональными литералами (*Function Literal*);

– Использование синтаксиса стрелочных функций (*Arrow Function*).

Примеры объявления функций такими способами:

```
function doSomething(){}
let doSomething = function() {}
let doSomething = () = > {};
```

Кроме того, функции можно вызывать различными способами: обычным, в виде метода объекта, в виде конструктора, в виде функции с использованием метода *apply*(), в виде функции с использованием метода *bind*().

Функции можно вызывать с большим или меньшим количеством аргументов, чем то количество параметров, которое было задано при их объявлении. В ходе работы функции «лишние» аргументы будут просто проигнорированы (хотя у функции будет доступ к ним), отсутствующие параметры получат значение undefined.

У функций есть два псевдо-параметра: this и arguments.

Ключевое слово this представляет собой контекст функции. Значение, на которое оно указывает, зависит от того, как была вызвана функция.

Ключевое слово arguments — это псевдопараметр, который даёт доступ ко всем аргументам, использованным при вызове функции. Он похож на массив, но массивом не является. В частности, у него нет методов массива.

2.3.9 Для работы с датой и временем в языке *JavaScript* используются объекты *Date*. При создании объекта без указания параметров он инициализируется текущей датой и временем.

Объекты Date имеют множество методов для получения и установки даты и времени, их преобразования к различным единицам измерения времени, вычисления разности дат и некоторых других операций. Также присутствует механизм авто-исправления некорректных параметров конструктора для даты, например, когда 32-й день месяца преобразуется к первому дню следующего.

2.3.10Метод глобального объекта *eval*() выполняет *JavaScript*-код, представленный строкой-параметром. Метод часто используется для вычисления поступающих арифметических выражений. Однако перед его использованием всегда нужно убедиться, что в поступающей на вход строке не содержится фрагмент кода, который могут использовать злоумышленники.

2.3.11 *JavaScript* является языком с динамической типизацией. Это означает, что конкретные значения имеют типы, а переменные — нет. Во время выполнения программы в одну и ту же переменную можно записывать значения разных типов.

Для выяснения типа данных, хранящихся в переменной, можно использовать оператор *typeof*().

2.3.12 Среда выполнения *JavaScript* является однопоточной. Это, в частности, выражается в невозможности одновременного выполнения двух функций (если не учитывать возможности асинхронного выполнения кода,

которые мы тут не затрагиваем). В среде выполнения имеется так называемая очередь событий (*Event Queue*), хранящая список заданий, которые нужно обработать. Как результат, для однопоточной схемы выполнения *JS* несвойственна проблема взаимных блокировок ресурсов, поэтому тут не нужен механизм блокировок. Однако, код, попадающий в очередь событий, должен выполняться быстро. Если перегрузить тяжёлой работой, в браузерном приложении, главный поток, страница приложения не будет реагировать на воздействия пользователя и браузер предложит закрыть эту страницу.

2.3.13 В JavaScript имеется механизм для обработки исключений. Работает он по вполне обычному для подобных механизмов принципу: код, который может вызвать ошибку, оформляют с использованием конструкции *try*/*catch*. Сам код находится в блоке *try*, ошибки обрабатываются в блоке *catch*.

2.3.14 В JavaScript функции являются объектами первого класса, язык поддерживает механизм замыканий. Это открывает путь к реализации методик функционального программирования в *JS*. В частности, речь идёт о возможности применения функций высшего порядка.

Замыкание — это внутренняя функция, у которой есть доступ к переменным, объявленным внутри родительской функции, даже после выполнения родительской функции.

Функция высшего порядка — это функция, которая способна принимать другие функции в качестве аргументов, возвращать функции, или делать и то и другое.

2.3.15 Для внедрения сценариев в HTML-документ имеются различные подходы:

– помещение кода непосредственно в атрибут события *HTML*-элемента;

– помещение кода внутри тега <*script*>;

– помещение скриптов во внешний файл (с расширением *.js*), а затем связать его с документом *HTML*.

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Обновление структуры проекта

Для выполнения лабораторной работы были созданы три новые страницы: *ActionsExamplePage*, *CalculatorPage*, *SignUpPage*. В соответствующих им директориях расположены файлы *index.html*, *index.js*, *styles.css*. Полученная структура проекта приведена на рисунке 2.

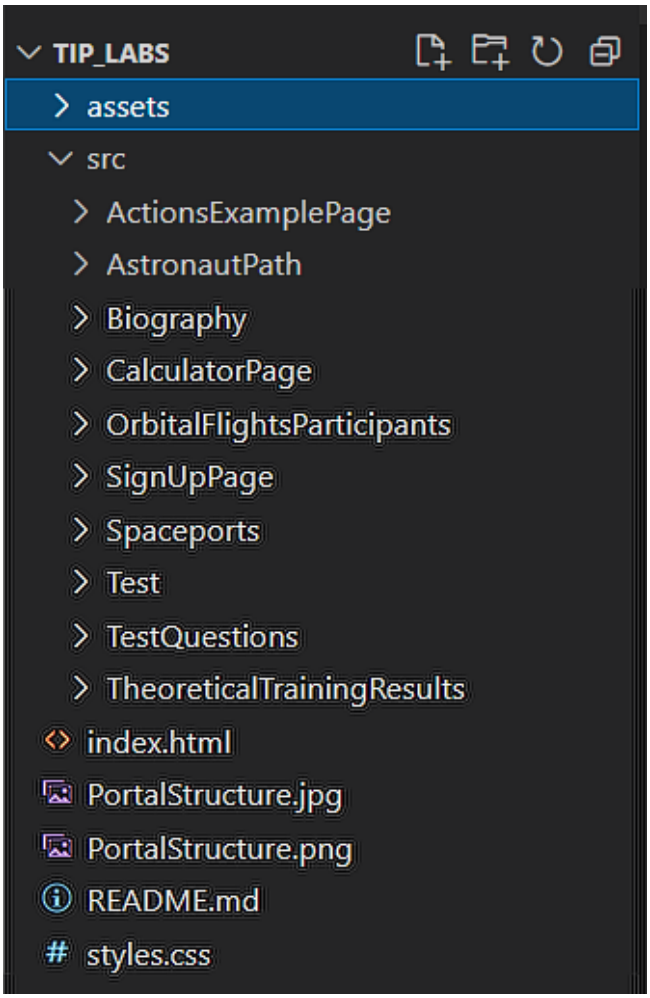

Рисунок 2 – Обновлённая структура проекта

Связи между *html*-документами и соответствующими им локальными *JavaScript*-файлами устанавливались через элемент *script* следующим образом:

<script type="text/javascript" src="./index.js"></script> .

3.2 Руководство пользователя

3.2.1 После доработки сайта приведём описание новых страниц для пользователя.

На сайт были добавлены страницы со следующими названиями:

– «Примеры использования Alert, Confirm и Prompt»;

– «Калькулятор»;

– «Форма регистрации».

Осуществить переход на данные страницы можно по ссылкам из «домашней страницы».

На новых страницах сайта также присутствует панель навигации (расположена в «шапке» страниц), а для возврата на «домашнюю страницу» в «подвале» этих страниц присутствует соответствующая ссылка.

3.2.2 «Домашняя страница» содержит список ссылок на все информативные страницы сайта. Это первая страница, которая отображается пользователю при открытии сайта. Для перехода на другие страницы достаточно выбрать одну из ссылок в списке навигации.

Домашняя страница сайта представлена на рисунке 3.

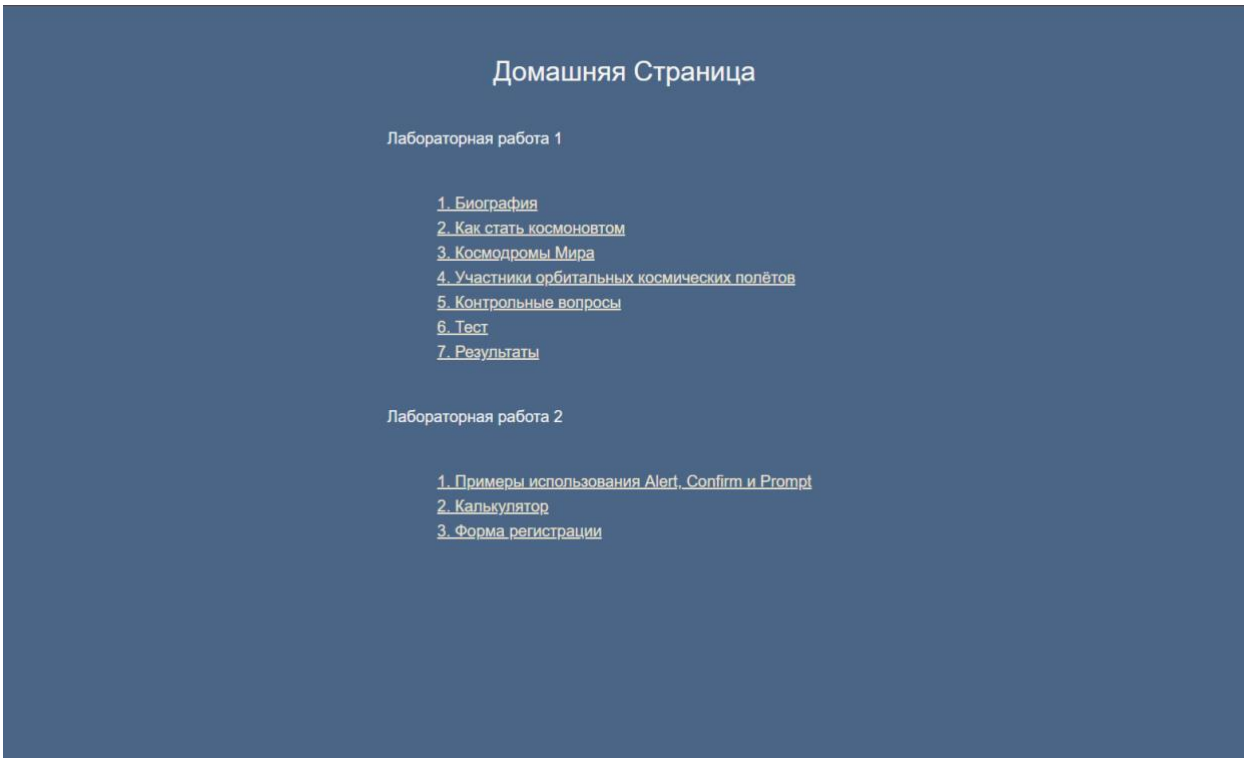

Рисунок 3 – «Домашняя страница»

3.2.3 Страница «Примеры использования *Alert*, *Confirm* и *Prompt*» содержит три кнопки для демонстрации взаимодействия сайта с пользователем посредством методов *alert*(), *confirm*() и *prompt*() объекта *window*.

В «шапке» страницы расположена панель навигации для дальнейших переходов по страницам сайта. В «подвале» страницы – ссылка для возврата на «домашнюю страницу».

Страница «Примеры использования *Alert*, *Confirm* и *Prompt*» приведена на рисунке 4.

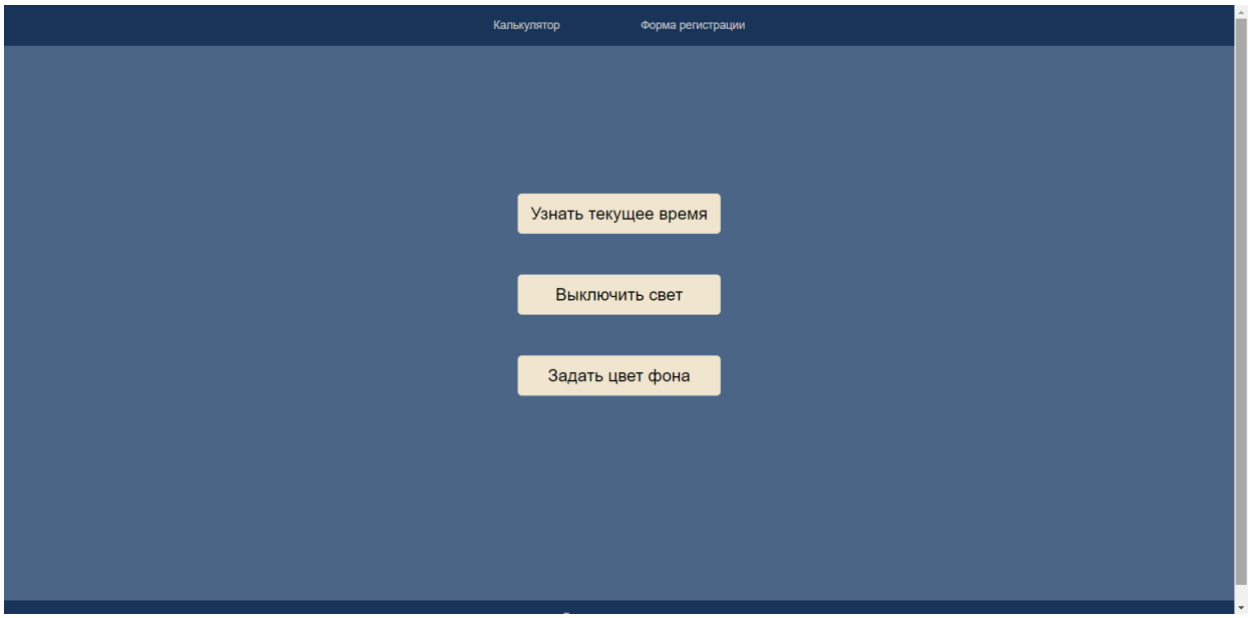

Рисунок 4 – Страница «Примеры использования Alert, Confirm и Prompt»

Взаимодействие страницы с пользователем показано на рисунках 5-7.

При нажатии на кнопку «Узнать текущее время» вверху страницы отображается модальное окно с локальным временем и кнопкой подтверждения «ОК». Результат представлен на рисунке 5.

При нажатии на кнопку «Выключить свет» вверху страницы отображается модальное окно с запросом подтверждения действия и кнопками «ОК» и «Отмена». При утвердительном ответе фон страницы изменяется на тёмный. Повторное нажатие на уже переименованную кнопку «Включить свет» отобразит новое модальное окно с запросом аналогичного подтверждения. При утвердительном ответе возвращается исходный фон страницы. Результат отображён на рисунке 6.

При нажатии на кнопку «Задать цвет фона» вверху страницы отображается модальное окно с запросом ввода нового фонового цвета в шестнадцатиричном формате (полном или сокращённом, например, *#FF0000* эквивалентен *#F00*). При вводе корректного значения цвет фона изменяется на указанный при вводе, иначе – появляется модальное окно с сообщением о некорректном вводе. Результат показан на рисунке 7.

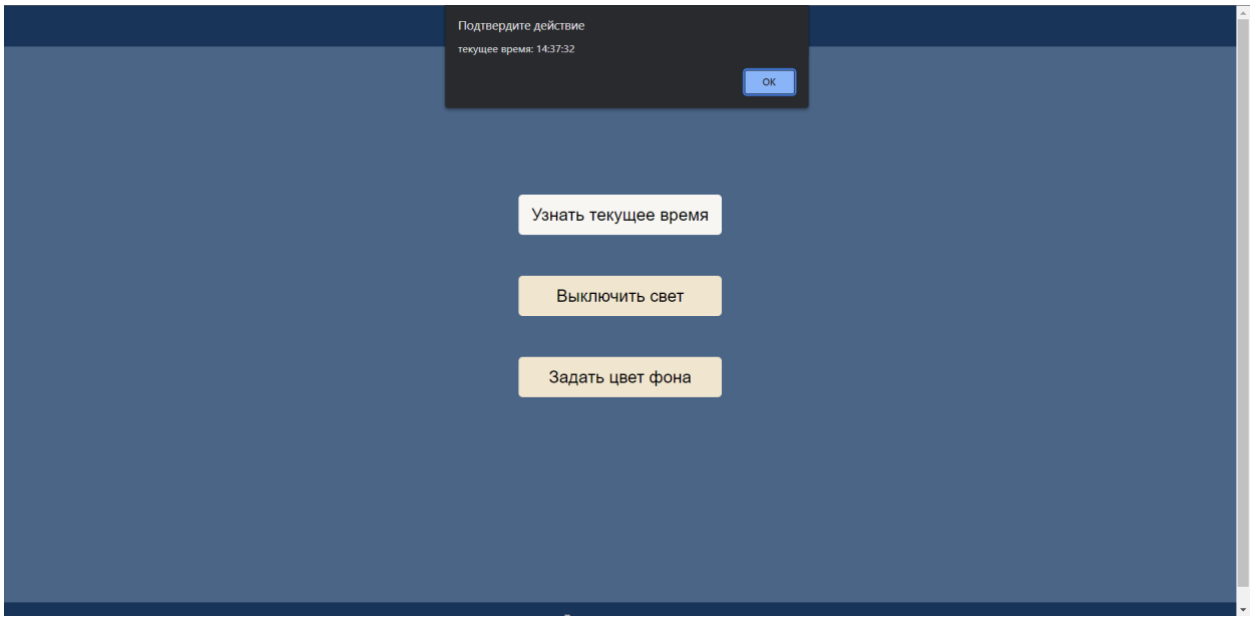

Рисунок 5 –Использование *alert*() для информирования пользователя

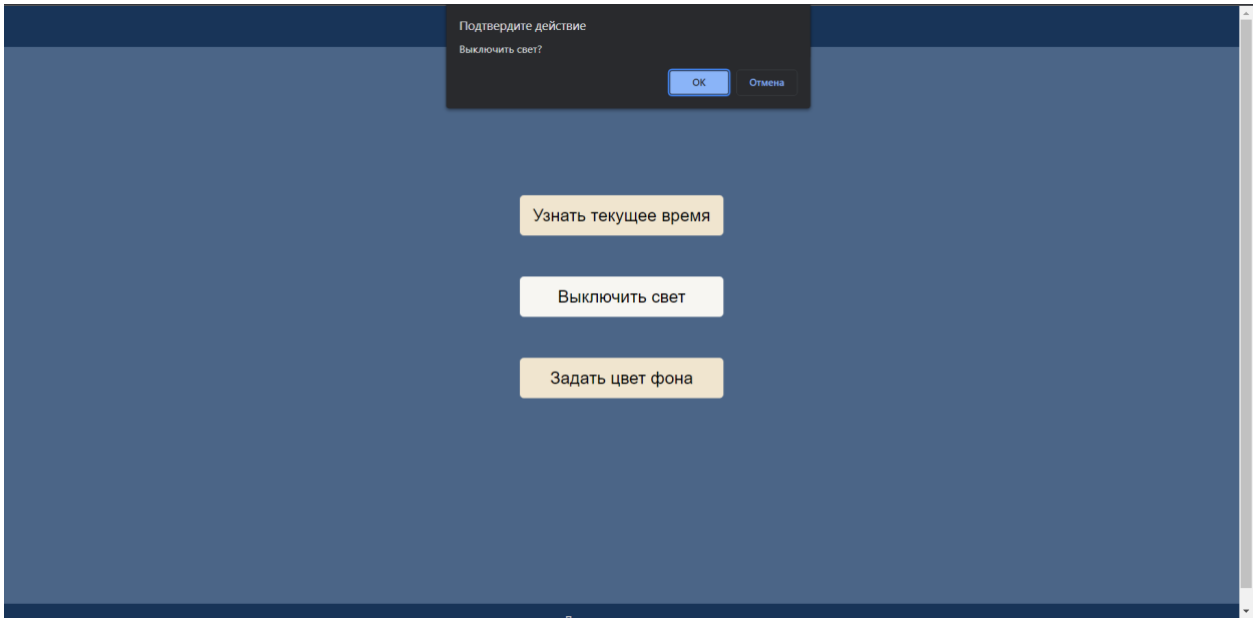

Рисунок 6 – Использование *confirm*() для запроса подтверждения действия от пользователя

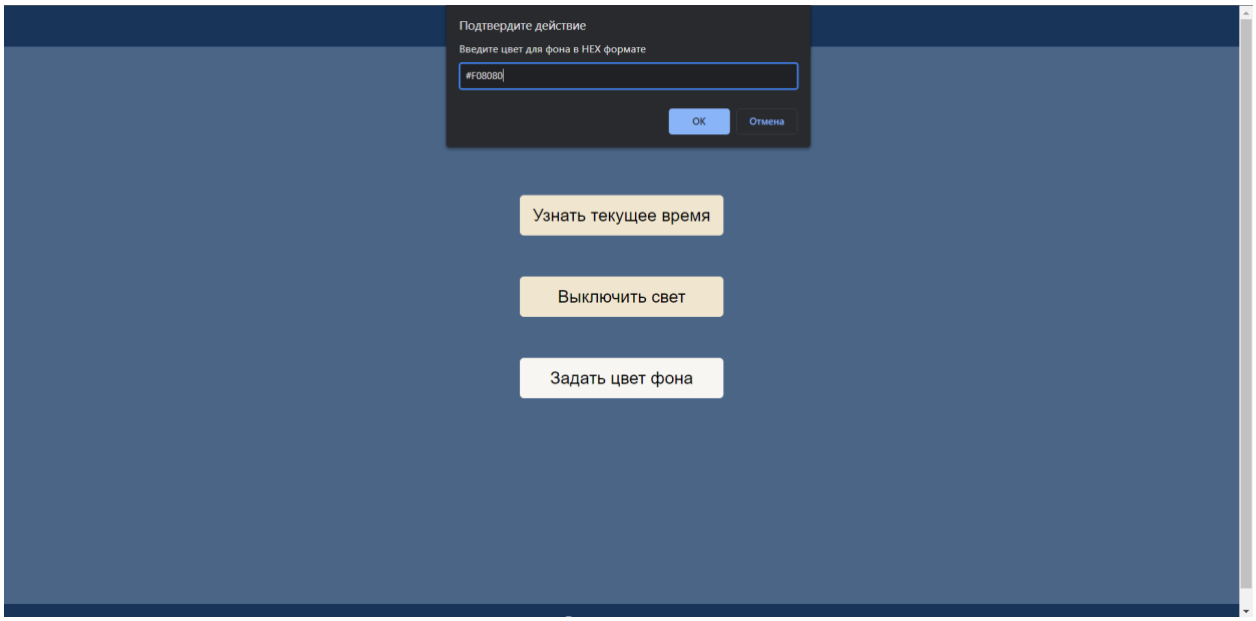

Рисунок 7 – Использование prompt() для запроса ввода данных пользователем

3.2.4 Страница «Калькулятор» содержит поле для ввода строки с вычислимым математическим выражением, поле для вывода результата и кнопку «Рассчитать». В «шапке» страницы расположена панель навигации для дальнейших переходов по страницам сайта. В «подвале» страницы – ссылка для возврата на «домашнюю страницу».

Созданная страница представлена на рисунке 8.

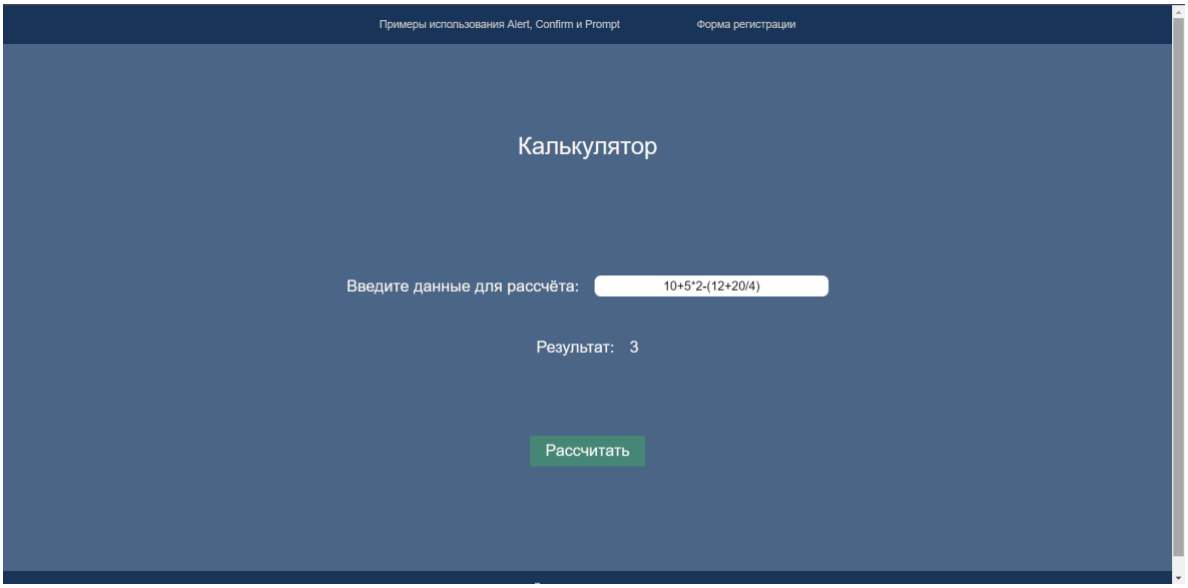

Рисунок 8 – Страница «Калькулятор»

Калькулятор поддерживает следующие математические операции: сложение («+»), вычитание («-»), умножение («*»), деление («/»), нахождение остатка от деления («%»), постфиксные и префиксные инкременты и декременты(«++», «--»).

При вводе корректного и вычислимого математического выражения его результат будет вычислен и отображён в поле результата. При вводе некорректного выражения в поле результата будет выведено соответствующее сообщение (рисунок 9).

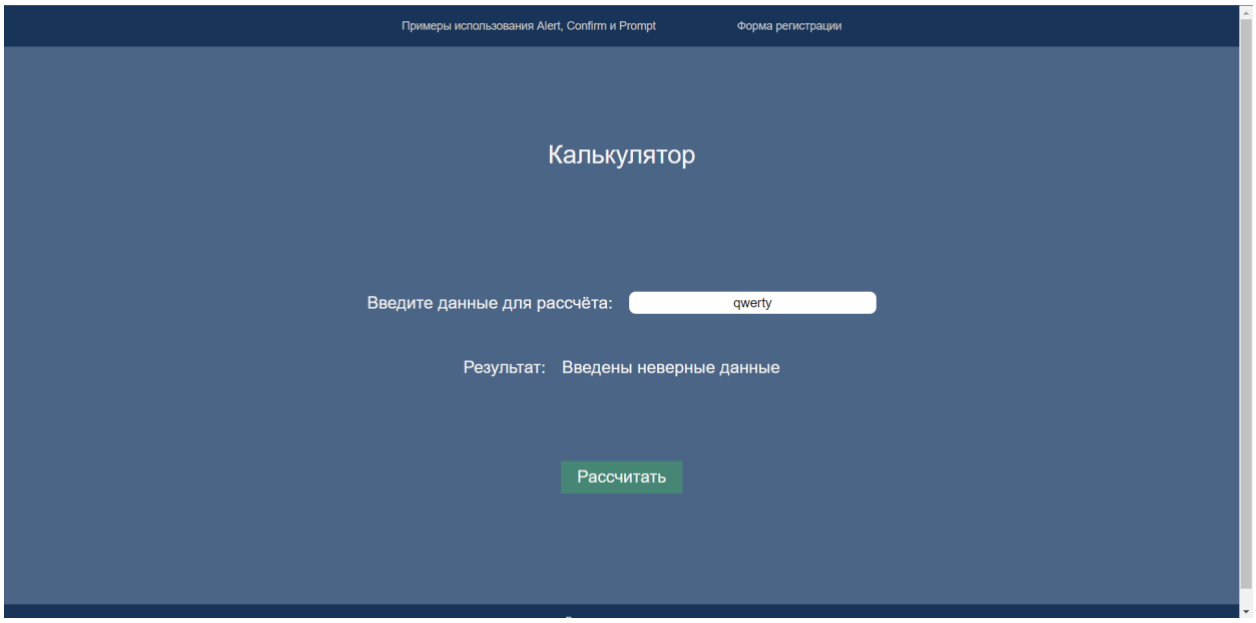

Рисунок 9 – Ввод некорректного математического выражения на странице «Калькулятор»

3.2.5 Страница «Форма регистрации» содержит форму регистрации для пользователей, в которой необходимо указать имя, фамилию и дату рождения. Язык формы можно выбрать в выпадающем списке, расположенном в «шапке» страницы. По умолчанию выбран русский язык.

Как и на предыдущих страницах, в «шапке» страницы расположена панель навигации для дальнейших переходов по страницам сайта, а в «подвале» страницы – ссылка для возврата на «домашнюю страницу».

Полученная страница на русском и английском языках представлена на рисунке 10 и рисунке 11 соответственно.

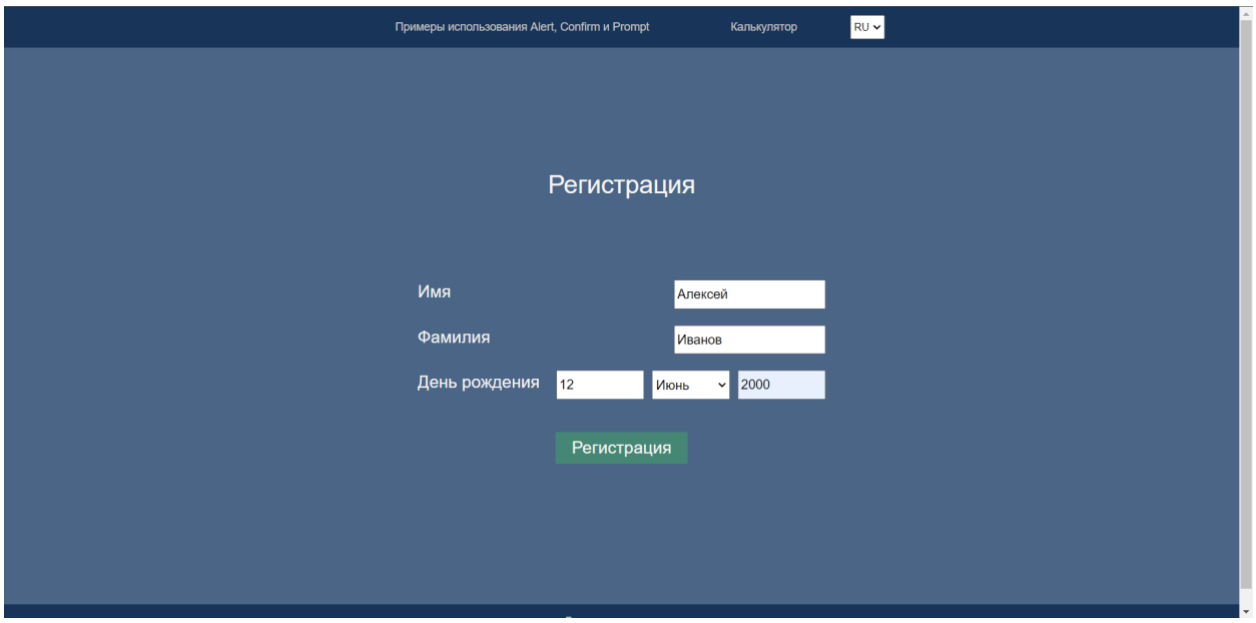

Рисунок 10 – Страница «Форма регистрации» на русском языке

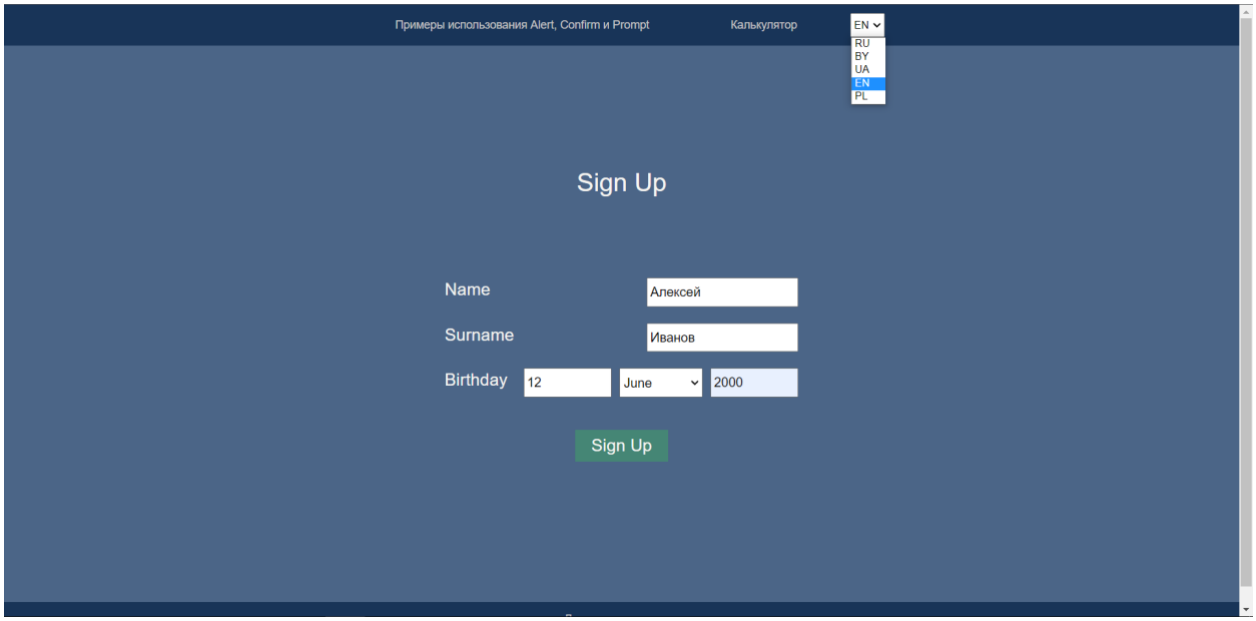

Рисунок 11 – Страница «Форма регистрации» на английском языке

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы закрепил теоретические знания основ языка программирования JavaScript, а также получил дополнительные практические навыки его применения для создания динамических веб-страниц.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Флэнаган, Дэвид *JavaScript*. Подробное руководство / Дэвид Флэнаган. – М.: Питер, 2021. – 720 с.

[2] Крокфорд, Дуглас Как устроен *JavaScript* / Дуглас Крокфорд – СПб. : Питер, 2019. – 304 с.

[3] Справочник по *HTML* и *CSS* [Электронный ресурс]. – Режим доступа : *http*://*htmlbook.ru*/.

[4] *HTML Tutorial* [Электронный ресурс]. – Режим доступа : *https://www.w3schools.com/html/*.

ПРИЛОЖЕНИЕ А
(СПРАВОЧНОЕ)
ЛИСТИНГ КОДА СТРАНИЦ

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./styles.css">
    <title>Калькулятор</title>
</head>

<body>
    <header class="header">
        <nav>
            <a href="../ActionsExamplePage/index.html">Примеры использования Alert, Confirm и
                Prompt</a>
            <a href="../SignUpPage/index.html">Форма регистрации</a>
        </nav>
    </header>

    <main>
        <section class="calculator">
            <h1>Калькулятор</h1>

            <div class="calculator-inputs">
                <label for="data-to-calculate" class="data-to-calculate-label">Введите данные для рассчёта:</label>
                <input id="data-to-calculate" class="data-to-calculate">
            </div>

            <div class="calculator-result">
                <h2 class="result-title">Результат:</h2>
                <span class="result">0</span>
            </div>

            <button class="calculate-button">Рассчитать</button>
        </section>
    </main>

    <footer>
        <a href="../../index.html" class="home-page-link">Домашнаяя страница<a>
    </footer>
</body>

<script type="text/javascript" src="./index.js"></script>

</html>
```

Рисунок А.1 – Исходный код страницы CalculatorPage/index.html

```
47: const NUMBER_OR_DOT_REGEXP = /[0-9]|\./gi
48:
49: function setup() {
50:     const [result] = document.getElementsByClassName("result");
51:     const [dataToCalculateInput] = document.getElementsByClassName("data-to-
    calculate");
52:     const [calculateButton] = document.getElementsByClassName("calculate-button");
53:
54:     calculateButton.onclick = () => {
55:         const inputValue = dataToCalculateInput.value;
56:         // to disable code executing in eval() except calculations
57:         const safeDataToCalculate = inputValue.replace(/[^-()\d/*+%. ]/g, '');
58:
59:         const dataWithReplacedPrefixAndPostifixOperatorsArray = [];
60:
61:         for (let i = 0; i < safeDataToCalculate.length; i++) {
62:             if (safeDataToCalculate[i] === '+' || safeDataToCalculate[i] === '-') {
63:                 const operator = safeDataToCalculate[i] === '+' ? '+' : '-';
64:
65:                 if (safeDataToCalculate[i+1] === operator) {
66:                     if (safeDataToCalculate[i-1]?.search(NUMBER_OR_DOT_REGEXP) > -1) {
67:                         // postfix operator
68:
69:                         let j = i-1; //last digit pos
70:                         let number = '';
71:
72:                         while(safeDataToCalculate[j]?.search(NUMBER_OR_DOT_REGEXP) > -
    1) {
73:                             number =
    dataWithReplacedPrefixAndPostifixOperatorsArray.pop() + number;
74:                             j--;
75:                         }
76:
77:
    dataWithReplacedPrefixAndPostifixOperatorsArray.push(`(${number}${operator}1)`);
78:                         i += 1;
79:                     } else {
80:                         // prefix operator
81:
82:                         let j = i+2; //first digit pos
83:                         let number = '';
84:
85:                         while(safeDataToCalculate[j]?.search(NUMBER_OR_DOT_REGEXP) > -
    1) {
86:                             number += safeDataToCalculate[j];
87:                             j++;
88:                         }
89:
90:
    dataWithReplacedPrefixAndPostifixOperatorsArray.push(`(${number}${operator}1)`);
91:
92:                         i += (number.length + 1);
93:                     }
94:                 } else {
95:
    dataWithReplacedPrefixAndPostifixOperatorsArray.push(safeDataToCalculate[i]);
96:                 }
97:             } else {
98:
    dataWithReplacedPrefixAndPostifixOperatorsArray.push(safeDataToCalculate[i]);
99:             }
100:        }
101:
102:        const dataToCalculate =
    dataWithReplacedPrefixAndPostifixOperatorsArray.join('');
103:
104:        try {
```

```
            const calculationResult = eval(dataToCalculate);

            if ( typeof calculationResult !== undefined
                && calculationResult !== null
                && !isNaN(parseInt(calculationResult))) {
                    result.textContent = calculationResult;
                } else {
                    result.textContent = "Введены неверные данные";
                }
        } catch(err) {
            console.error(err);
            result.textContent = "Введены неверные данные";
        }
    };

    dataToCalculateInput.addEventListener("keyup", function(event) {
        // Number 13 is the "Enter" key on the keyboard
        if (event.keyCode === 13) {
        calculateButton.click();
        }
    });
}

setup();
```

Рисунок А.2 – Исходный код файла CalculatorPage/index.js

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./styles.css">
    <title>Alert, Confirm, Prompt</title>
</head>

<body>
    <header class="header">
        <nav>
            <a href="../CalculatorPage/index.html">Калькулятор</a>
            <a href="../SignUpPage/index.html">Форма регистрации</a>
        </nav>
    </header>

    <main>
        <section class="actions">
            <button class="alertButton">Узнать текущее время</button>
            <button class="confirmButton">Выключить свет</button>
            <button class="promptButton">Задать цвет фона</button>
        </section>
    </main>

    <footer>
        <a href="../../index.html" class="home-page-link">Домашнаяя страница<a>
    </footer>
</body>

<script type="text/javascript" src="./index.js"></script>

</html>

</html>
```

Рисунок А.3 – Исходный код страницы ActionsExamplePage/index.html

```
166:
167: const LIGHT_COLOR = "#4B6587";
168: const DARK_COLOR = "#000000";
169:
170: function setup() {
171:     const [body] = document.getElementsByTagName("body");
172:
173:     const [alertButton] = document.getElementsByClassName("alertButton");
174:
175:     alertButton.onclick = () => {
176:         const currentTime = new Date();
177:         alert(`текущее время: ${currentTime.toLocaleTimeString()}`);
178:     };
179:
180:     const [confirmButton] = document.getElementsByClassName("confirmButton");
181:     let isLightOn = true;
182:
183:     confirmButton.onclick = () => {
184:         if (isLightOn) {
185:             const result = confirm("Выключить свет?");
186:
187:             if (result) {
188:                 body.style.backgroundColor = DARK_COLOR;
189:                 isLightOn = false;
190:                 confirmButton.textContent = "Включить свет";
191:             }
192:         } else {
193:             const result = confirm("Включить свет?");
194:
195:             if (result) {
196:                 body.style.backgroundColor = LIGHT_COLOR;
197:                 isLightOn = true;
198:                 confirmButton.textContent = "Выключить свет";
199:             }
200:         }
201:     };
202:
203:     const [promptButton] = document.getElementsByClassName("promptButton");
204:
205:     promptButton.onclick = () => {
206:         let color = prompt("Введите цвет для фона в HEX формате");
207:
208:         if (!color.includes('#')) {
209:             color = `#${color}`;
210:         }
211:
212:         if (/^#([0-9A-F]{3}){1,2}$/i.test(color)) {
213:             body.style.backgroundColor = color;
214:         } else {
215:             alert('Невеный формат цвета! Введите цвет в HEX формате.');
216:         }
217:     };
218: };
219:
220: setup();
```

Рисунок А.4 – Исходный код страницы ActionsExamplePage/index.js

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./styles.css">
    <title>Регистрация</title>
</head>

<body>
    <header class="header">
        <nav>
            <a href="../ActionsExamplePage/index.html">Примеры использования Alert, Confirm и
                Prompt</a>
            <a href="../CalculatorPage/index.html">Калькулятор</a>
        </nav>
        <select class="local-select"></select>
    </header>

    <main>
        <section class="sign-up">
            <h1 class="sign-up-title"></h1>

            <form class="sign-up-form">
                <div class="input-container">
                    <label class="label name-label" for="name"></label>
                    <input type="text" id="name">
                </div>

                <div class="input-container">
                    <label class="label surname-label" for="surname"></label>
                    <input type="text" id="surname">
                </div>

                <div class="input-container">
                    <label class="label birth-day-label" for="birth-day-input"></label>

                    <input id="birth-day-input" type="text" pattern="^[ 0-9]+$">
                    <select class="birth-month-select"></select>
                    <input id="birth-year-input" type="text" pattern="^[ 0-9]+$">
                </div>

                <button class="submit-button"></button>
            </form>
        </section>
    </main>

    <footer>
        <a href="../../index.html" class="home-page-link">Домашнаяя страница<a>
    </footer>
</body>

<script type="text/javascript" src="./index.js"></script>

</html>
```

Рисунок А.5 – Исходный код страницы SignUpPage/in/index.html

```
function setup() {
    const LOCAL_STORAGE_LOCALE = "LOCAL_STORAGE_LOCALE";

    const localeValues = {
        "RU": {
            months: ["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь", "Июль", "Август",
            "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"],
            name: "Имя",
            surname: "Фамилия",
            birthday: "День рождения",
            signUpTitle: "Регистрация",
        },
        "BY": {
            months: ["Студзень", "Люты", "Сакавік", "Красавік", "Май", "Червень", "Ліпень", "Жнівень",
            "Верасень", "Кастрычнік", "Лістапад", "Снежань"],
            name: "Імя",
            surname: "Прозвішча",
            birthday: "Дзень нараджэння",
            signUpTitle: "Рэгістрацыя",
        },
        "UA":  {
            months: ["Січень", "Лютий", "Березень", "Квітень", "Травень", "Червень", "Липень", "Серпень",
            "Вересень", "Жовтень", "Листопад", "Грудень"],
            name: "Ім'я",
            surname: "Прізвище",
            birthday: "День народження",
            signUpTitle: "Реєстрація",
        },
        "EN": {
            months: ["January", "February", "March", "April", "May", "June", "July", "August",
            "September", "October", "November", "December"],
            name: "Name",
            surname: "Surname",
            birthday: "Birthday",
            signUpTitle: "Sign Up",
        },
        "PL": {
            months: ["Styczeń", "Luty", "Marzec", "Kwiecień", "Maj", "Czerwiec", "Lipiec", "Sierpień",
            "Wrzesień", "Październik", "Listopad", "Grudzień"],
            name: "Nazwa",
            surname: "Nazwisko",
            birthday: "Urodziny",
            signUpTitle: "Rejestracja",
        },
    }

    const [localSelect] = document.getElementsByClassName("local-select");
    const locals = Object.keys(localeValues);

    locals.forEach((locale) => {
        const localOption = document.createElement("option");
        localOption.innerText = locale;

        localSelect.append(localOption);
    });

    const [signUpTitle] = document.getElementsByClassName("sign-up-title");
    const [nameLabel] = document.getElementsByClassName("name-label");
    const [surnameLabel] = document.getElementsByClassName("surname-label");
    const [birthMonthSelect] = document.getElementsByClassName("birth-month-select");
    const [birthdayLabel] = document.getElementsByClassName("birth-day-label");
    const [submitButton] = document.getElementsByClassName(" submit-button");
```

```
340:
341:     localSelect.onchange = (event) => {
342:         renderLocale(event.target.value);
343:         localStorage.setItem(LOCAL_STORAGE_LOCALE, event.target.value);
344:     };
345:
346:     const defaultLocale =  localStorage.getItem(LOCAL_STORAGE_LOCALE) ?? locals[0];
347:
348:     localSelect.value = defaultLocale;
349:     renderLocale(localSelect.value);
350:
351:     function renderLocale(locale) {
352:         const values = localeValues[locale];
353:
354:         if (values) {
355:             nameLabel.textContent = values.name;
356:             surnameLabel.textContent = values.surname;
357:             signUpTitle.textContent = values.signUpTitle;
358:             submitButton.textContent = values.signUpTitle;
359:             birthdayLabel.textContent = values.birthday;
360:             document.title = values.signUpTitle;
361:
362:             birthMonthSelect.innerHTML = "";
363:
364:             values.months.forEach((month) => {
365:                 const monthOption = document.createElement("option");
366:                 monthOption.innerText = month;
367:
368:                 birthMonthSelect.append(monthOption);
369:             });
370:         }
371:     }
372: }
373:
374: setup();
```

Рисунок А.6 – Исходный код страницы SignUpPage /index.js