**федеральное государственное автономное образовательное учреждение высшего образования**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**(ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ)**

**(Факультет информационных технологий)**

***(Институт Принтмедиа и информационных технологий)***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА № 5**

**Дисциплина:** Инфокоммуникационные системы и сети

**Тема:** "Фоновый обмен данными в ИС"

**Выполнил(а): студент(ка) группы 221-3711**

Морозов Константин Алексеевич

(Фамилия И.О.)

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_***

(Фамилия И.О., степень, звание) **(Оценка)**

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Замечания:**

**Москва2024**

Лабораторной работы № 5

Фоновый обмен данными в ИС

Задача: Разработайте веб-приложение «Калькулятор», выполняющий следующие операции:

Сложение.

Вычитание.

Умножение.

Деление.

Возведение в n-ую степень.

Корень n-ой степени.

Синус.

Косинус.

Тангенс.

Котангенс.

Форма должна содержать два элемента: для ввода выражения и отображения результата. Форма не должна содержать кнопок. Результат, вычисляемый в соответствии с алгоритмом, реализуемым на сервере, в процессе ввода операндов и операций в одном элементе ввода, должен отображаться в элементе формы, предназначенном для отображения результата.

Ссылка на git: https://github.com/Sollimba/CalculatorApp

### Теоретическая часть

технология **AJAX** (Asynchronous JavaScript and XML), позволяющая выполнять асинхронные запросы между браузером и веб-сервером. AJAX является независимой от программного обеспечения веб-сервера и использует следующие веб-стандарты:

* **JavaScript** — язык программирования, обеспечивающий функциональность веб-приложений.
* **XML** (или JSON) — форматы обмена данными, используемые для передачи информации между клиентом и сервером.
* **HTML** — язык разметки, отвечающий за структуру и содержание веб-страниц.
* **CSS** — каскадные таблицы стилей, определяющие внешний вид веб-страниц.

Благодаря поддержке этих стандартов всеми основными веб-браузерами, AJAX-приложения являются кроссбраузерными и платформо-независимыми. Основное преимущество AJAX заключается в том, что вместо полной перезагрузки страницы можно обновить только изменившуюся часть, что значительно сокращает объем передаваемых данных и ускоряет взаимодействие с пользователем. Однако AJAX также имеет недостатки, такие как необходимость включения JavaScript в браузере и возможные проблемы с производительностью при большом количестве запросов.

### Важнейшие компоненты AJAX

Ключевым элементом технологии AJAX является объект **XMLHttpRequest**, который поддерживается всеми современными браузерами. Объекты этого класса позволяют отправлять HTTP-запросы на сервер и обрабатывать ответы. Создание экземпляров XMLHttpRequest отличается в зависимости от браузера:

* Для Internet Explorer 5.5+ используется:

var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");

* Для остальных браузеров:

var xmlHttp = new XMLHttpRequest();

#### Методы и свойства XMLHttpRequest

1. **Методы**:
   * **open()** — инициализирует новый HTTP-запрос.
   * **send()** — отправляет запрос на сервер.
   * **abort()** — отменяет текущий запрос.
2. **Свойства**:
   * **onreadystatechange** — устанавливает обработчик, который срабатывает при каждом изменении состояния объекта.
   * **readyState** — определяет текущее состояние объекта (от 0 до 4).
   * **responseText** — содержит текстовый ответ сервера.

Состояние readyState имеет следующие значения:

* 0: Запрос не инициализирован.
* 1: Запрос создан.
* 2: Запрос отправлен.
* 3: Запрос обрабатывается.
* 4: Запрос завершен, и данные ответа готовы к использованию.

### Библиотека jQuery

**jQuery** — это популярная библиотека JavaScript, которая упрощает взаимодействие с HTML-документами и работу с AJAX. jQuery позволяет легко управлять элементами DOM и манипулировать их атрибутами и содержимым, а также упрощает выполнение AJAX-запросов. Основной метод для работы с AJAX в jQuery — это $.ajax(), который предоставляет множество опций для настройки запросов.

#### Пример использования jQuery для AJAX

|  |
| --- |
| $.ajax({  type: "POST", // Метод передачи данных (GET или POST)  url: "/api/calculator", // URL-адрес для отправки запроса  data: JSON.stringify({ // Данные для отправки на сервер  FirstOperand: firstOperand,  Operation: operation,  SecondOperand: secondOperand  }),  contentType: 'application/json; charset=utf-8', // Тип содержимого  dataType: 'json', // Ожидаемый тип данных от сервера  success: function(response) { // Обработка успешного ответа  $('#result').text('Result: ' + response);  },  error: function(xhr) { // Обработка ошибки  $('#result').text('Error: ' + xhr.responseText);  }  }); |

### **Ход работы**

1. **Планирование и проектирование**:

### Описание пользовательских функций и алгоритмов

1. **Ввод данных**: Пользователь вводит выражение (например, "2+3" или "sin30") в текстовое поле.
2. **Обработка ввода**:
   * При нажатии клавиши Enter вызывается обработчик событий.
   * Введенное выражение проверяется на соответствие регулярным выражениям для двух операндов или одной операции.
3. **Формирование запроса**:
   * Если выражение соответствует формату, формируется AJAX-запрос с данными: первый операнд, операция, второй операнд (если есть).
4. **Обработка ответа**:
   * Если запрос успешен, результат вычисления отображается на странице.
   * В случае ошибки выводится сообщение об ошибке.
5. **Создание модели данных**: Для передачи данных между клиентом и сервером создан класс CalculatorModel, который содержит свойства FirstOperand, Operation и SecondOperand:

|  |
| --- |
| namespace LabCalculator.Models  {  public class CalculatorModel  {  public double FirstOperand { get; set; }  public required string Operation { get; set; }  public double? SecondOperand { get; set; }  }  } |

Изменение свойства SecondOperand на Nullable позволило обрабатывать случаи, когда операция не требует второго операнда (например, для функций sqrt, sin, cos и т. д.).

1. **Реализация серверной логики**: В классе CalculatorController реализована логика обработки запросов. При получении данных выполняется соответствующая математическая операция:

|  |
| --- |
| using Microsoft.AspNetCore.Mvc;  using LabCalculator.Models;  [Route("api/[controller]")]  [ApiController]  public class CalculatorController : ControllerBase  {  [HttpPost]  public ActionResult<double> Calculate([FromBody] CalculatorModel model)  {  // Проверка на наличие первого операнда и операции  if (model.FirstOperand == 0 && string.IsNullOrEmpty(model.Operation))  {  return BadRequest("Model is required.");  }  double result = 0;  switch (model.Operation)  {  case "+":  result = model.FirstOperand + model.SecondOperand.GetValueOrDefault();  break;  case "-":  result = model.FirstOperand - model.SecondOperand.GetValueOrDefault();  break;  case "\*":  result = model.FirstOperand \* model.SecondOperand.GetValueOrDefault();  break;  case "/":  if (model.SecondOperand == 0) return BadRequest("Division by zero.");  result = model.FirstOperand / model.SecondOperand.GetValueOrDefault();  break;  case "^":  result = Math.Pow(model.FirstOperand, model.SecondOperand.GetValueOrDefault());  break;  case "sqrt":  result = Math.Sqrt(model.FirstOperand);  break;  case "sin":  result = Math.Sin(model.FirstOperand \* (Math.PI / 180)); // Преобразование в радианы  break;  case "cos":  result = Math.Cos(model.FirstOperand \* (Math.PI / 180)); // Преобразование в радианы  break;  case "tan":  result = Math.Tan(model.FirstOperand \* (Math.PI / 180)); // Преобразование в радианы  break;  case "cot":  result = 1 / Math.Tan(model.FirstOperand \* (Math.PI / 180)); // Преобразование в радианы  break;  default:  return BadRequest("Invalid operation");  }  // Округляем результат до одного знака после запятой  result = Math.Round(result, 1);  return Ok(result);  }  } |

Данная реализация учитывает различные математические операции и обрабатывает ошибки, такие как деление на ноль.

**Разработка клиентской части**: В HTML-документе создано поле ввода для выражений и блок для отображения результата. Использована jQuery для обработки ввода пользователя и отправки AJAX-запросов на сервер при нажатии клавиши Enter:

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8" />  <title>Calculator</title>  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>  <script>  $(document).ready(function () {  $('#expression').on('keypress', function (e) {  if (e.which === 13) { // Enter key  var input = $(this).val().trim(); // Удаляем пробелы в начале и в конце  var regex = /(-?\d+(\.\d+)?)\s\*([+\-\*/^])\s\*(-?\d+(\.\d+)?)/; // Для двух операндов с возможными пробелами  var singleArgRegex = /([+\-\*/^]|sqrt|sin|cos|tan|cot)\s\*(-?\d+(\.\d+)?)/; // Для одной операции с одним операндом  var match = input.match(regex) || input.match(singleArgRegex); // Попробовать найти два операнда или одну операцию  if (match) {  var firstOperand, operation, secondOperand = null; // Устанавливаем secondOperand в null по умолчанию  if (match.length === 6) { // Два операнда  firstOperand = parseFloat(match[1]);  operation = match[3];  secondOperand = parseFloat(match[4]);  } else { // Одна операция с одним операндом  firstOperand = parseFloat(match[2]); // Используем правильный индекс для первого операнда  operation = match[1];  }  // Выполняем AJAX-запрос с учетом операции  $.ajax({  type: 'POST',  url: '/api/calculator',  data: JSON.stringify({  FirstOperand: firstOperand,  Operation: operation,  SecondOperand: secondOperand // Это будет null, если в операции только один операнд  }),  contentType: 'application/json; charset=utf-8',  dataType: 'json',  success: function (response) {  $('#result').text('Result: ' + response);  },  error: function (xhr) {  $('#result').text('Error: ' + xhr.responseText);  }  });  } else {  $('#result').text('Error: Invalid input format. Use "operand operation operand"');  }  }  });  });  </script>  </head>  <body>  <h1>Calculator</h1>  <input type="text" id="expression" placeholder="Enter expression e.g. 2+3 or sin30" />  <div id="result"></div>  </body>  </html> |

В данном коде используется регулярное выражение для распознавания входного формата, позволяя пользователям вводить операции в различных форматах.

#### Структура и взаимодействие

1. **Тип запроса**:
   * type: "POST": Данный параметр указывает метод передачи данных. В данном случае используется метод POST, что означает, что клиент отправляет данные на сервер для обработки. Это подходит для операций, где необходимо передать информацию, которую сервер должен сохранить или обработать, например, вычисление математической операции.
2. **URL-адрес**:
   * url: "/api/calculator": Здесь определяется адрес, по которому сервер принимает запросы. Это относительный путь к API контроллеру, который обрабатывает математические операции. В архитектуре RESTful API этот адрес служит конечной точкой для взаимодействия между клиентом и сервером.
3. **Данные**:
   * data: JSON.stringify({...}): Данный параметр определяет данные, отправляемые на сервер. Они сериализуются в формат JSON, что обеспечивает удобный и стандартизированный способ передачи структурированных данных. Внутри объекта передаются три ключевых значения:
     + FirstOperand: первый операнд для математической операции.
     + Operation: строка, представляющая выполняемую операцию (например, сложение или вычитание).
     + SecondOperand: второй операнд, который может быть null, если операция требует только одного операнда.
4. **Тип содержимого**:
   * contentType: 'application/json; charset=utf-8': Указывает серверу, что данные, отправляемые клиентом, имеют тип JSON и используют кодировку UTF-8. Это позволяет серверу правильно интерпретировать входящие данные и обрабатывать их соответствующим образом.
5. **Ожидаемый тип данных**:
   * dataType: 'json': Этот параметр сообщает клиенту, что ожидается ответ в формате JSON от сервера. Это позволяет клиенту правильно обрабатывать и интерпретировать полученные данные.
6. **Обработка ответа**:
   * success: function(response) {...}: Эта функция обрабатывает успешный ответ от сервера. В случае успешного выполнения запроса (код состояния 200), функция получает данные (результат операции) и обновляет соответствующий элемент интерфейса (в данном случае #result), отображая пользователю результат вычисления.
7. **Обработка ошибок**:
   * error: function(xhr) {...}: Эта функция вызывается в случае ошибки при выполнении запроса. Например, если сервер возвращает код ошибки (например, 400 или 500), функция обрабатывает ответ и отображает сообщение об ошибке пользователю, обновляя элемент интерфейса, чтобы сообщить о возникшей проблеме.

**Проверка работы проекта**

После запуска проекта открывает <https://localhost:7173/> в браузере.

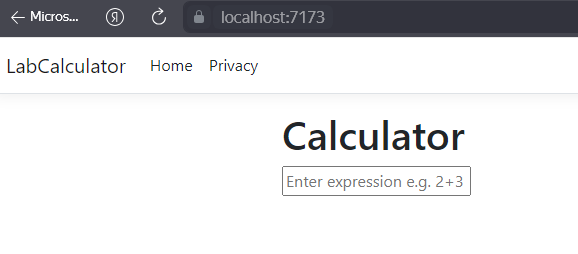


Рисунок 1

Проверяем работу каждого действия:

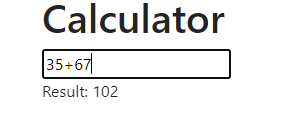


Рисунок 2

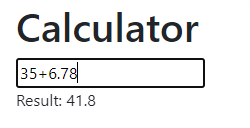


Рисунок 3

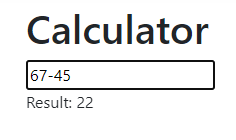


Рисунок 4

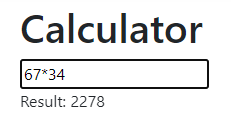


Рисунок 5

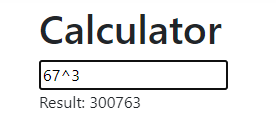


Рисунок 6

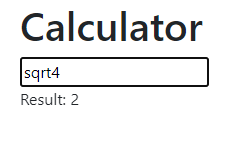


Рисунок 7

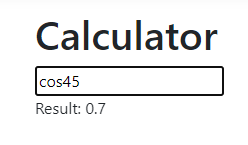


Рисунок 8

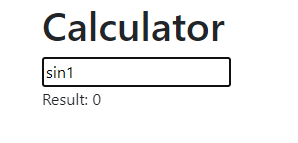


Рисунок 9

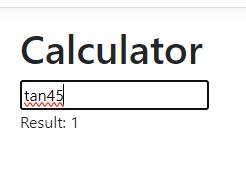


Рисунок 10

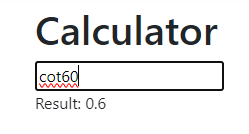


Рисунок 11