

Разработка веб-приложения на JavaScript для анализа и визуализации данных с использованием Chart.js

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель данной статьи – создание веб-приложения для анализа и визуализации данных. Для создания веб-приложения использовались язык программирования JavaScript для реализации логики приложения, библиотека Chart.js для создания интерактивных графиков, а также язык разметки HTML и для оформления дизайна CSS. В результате было создано веб-приложение, позволяющее пользователям загружать данные в форматах CSV и JSON, выбирать тип графика (столбчатая, линейная или круговая диаграмма) и визуализировать данные в удобной форме.

Ключевые слова: JavaScript, Chart.js, HTML, CSS, визуализация данных.

Developing a JavaScript web application for data analysis and visualization using Chart.js

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a web application for data analysis and visualization. To create the web application, the JavaScript programming language and the Chart library were used to implement the application logic.js for creating interactive graphs, as well as HTML markup language and CSS for design design. As a result, a web application was created that allows users to upload data in CSV and JSON formats, select a graph type (columnar, linear or pie chart) and visualize the data in a convenient form.

Keywords: JavaScript, Chart.js, HTML, CSS, data visualization.

1 Введение

1.1 Актуальность

В современном мире данные играют ключевую роль в принятии решений в различных сферах, включая бизнес, науку, образование и государственное управление. Однако raw-данные (сырые данные) часто сложны для восприятия и анализа без их визуализации. Графики и диаграммы позволяют преобразовать числовую информацию в наглядные образы, что упрощает интерпретацию данных и выявление закономерностей. В связи с

этим разработка инструментов для автоматизированной визуализации данных становится все более востребованной, особенно в условиях растущего объема информации и необходимости оперативного анализа.

JavaScript, как один из самых популярных языков программирования для веб-разработки, предоставляет широкие возможности для создания интерактивных и динамических веб-приложений. Библиотека Chart.js, построенная на основе JavaScript, позволяет легко интегрировать графики и диаграммы в веб-страницы, делая их интерактивными и адаптивными. Использование этих технологий для разработки приложений визуализации данных актуально, так как позволяет создавать легкие в использовании и доступные инструменты, которые могут быть развернуты на любом устройстве с веб-браузером.

Кроме того, поддержка популярных форматов данных, таких как CSV и JSON, делает такие приложения универсальными и применимыми в различных областях. Например, их можно использовать для анализа результатов опросов, визуализации статистических данных или мониторинга бизнес-показателей. Таким образом, разработка веб-приложения на JavaScript для анализа и визуализации данных с использованием Chart.js отвечает современным требованиям к обработке и представлению информации, что делает данную тему актуальной и востребованной.

1.2 Обзор исследований

Д.А. Андреевич, Я.В. Зиновьев, К.М. Кузьмин рассматривали существующие технологии и пакеты инструментов веб-разработки, способствующие быстрому и качественному созданию модулей графического представления и выгрузки данных [1]. Изучил библиотеки на языке JavaScript, позволяющие осуществлять визуализацию данных, представленных в различных форматах Я.С. Бизяркин [2]. О.А. Иванов и В. В. Вдовенко предложили решение проблемы в выборе средства визуализации большого количества данных в клиентском приложении, не используя flash объектов визуализации. [3]. В статье показал актуальность визуализации данных, рассматривал современные языки программирования, используемые для решения данной задачи. А.В. Ломаков 1 [4]. К.В. Рябинин рассматривал вопрос создания системы визуализации результатов научных расчётов. [5].

1.3 Цель исследования

Цель исследования -создание веб-приложения на JavaScript для анализа и визуализации данных с использованием языка программирования JavaScript, библиотеки Chart.js, языка гипертекстовой разметки HTML и языка описания внешнего вида документа CSS для оформления дизайна.

2 Материалы и методы

Для создания приложения используется язык программирования JavaScript, библиотека Chart.js, язык гипертекстовой разметки HTML, язык описания внешнего вида документа CSS. В качестве среды разработки

используется текстовым редактор Notepad++, оптимизированным для языков программирования. Работа будет происходить на готовых данных опроса csv формата и json формата которые можно скачать по ссылке:

https://drive.google.com/drive/folders/1Jw9W6ByfsJJPEcVNRg66g6J_xRvIAp20?usp=sharing

3 Результаты и обсуждение

Перед началом работы требуется скачать Notepad++ с официального сайта и установить.

Создадим файл index.html. Для этого необходимо открыть Notepad++, и для того, чтобы создать файл в меню необходимо выбрать файл -> новый (или нажмите Ctrl + N). В меню выбираем файл -> Сохранить как... В поле "Имя файла" введите название файла index с расширением .html, и сохраняем.

Для начала реализуем код с базовой структуры HTML-документа. HTML-документ начинается с объявления типа документа (<!DOCTYPE html>), что указывает браузеру, что это документ HTML. Внутри тега <html> задается язык страницы (lang="ru"), чтобы браузер корректно отображал текст на русском языке. В разделе <head> указывается кодировка UTF-8 (<meta charset="UTF-8">), которая поддерживает кириллицу и другие символы. Заголовок страницы (<title>) отображается в заголовке браузера. Для стилизации элементов подключается библиотека Bootstrap через CDN (<link>), а также добавляется блок <style> для кастомизации внешнего вида страницы. Для того, чтобы просмотреть файл index.html, необходимо нажать правой кнопкой мыши на файл, и выбрать открыть с помощью, и выбираем браузер.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Анализ данных</title>
  <link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css
" rel="stylesheet">
  <style>
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
      background-color: #f8f9fa;
    }
    .container {
      max-width: 1200px;
      margin-top: 50px;
    }
    #myChart {
      max-width: 1000px;
      height: 600px;
      margin: 0 auto;
    }
    .hidden {
      display: none;
    }
```

```
    }
    .button-group {
      margin-top: 20px;
      text-align: center;
    }
    .button-group button {
      margin: 5px;
    }
    .card {
      border: none;
      border-radius: 10px;
      box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    }
    .card-header {
      background-color: #007bff;
      color: white;
      border-radius: 10px 10px 0 0;
    }
    .form-control, .form-select {
      border-radius: 5px;
    }
    .btn-primary {
      background-color: #007bff;
      border: none;
      border-radius: 5px;
    }
    .btn-primary:hover {
      background-color: #0056b3;
    }
    .btn-secondary {
      background-color: #6c757d;
      border: none;
      border-radius: 5px;
    }
    .btn-secondary:hover {
      background-color: #5a6268;
    }
    .btn-success {
      background-color: #28a745;
      border: none;
      border-radius: 5px;
    }
    .btn-success:hover {
      background-color: #218838;
    }
  }
</style>
</head>
```

Далее прописываем основную структуру страницы. Внутри тега `<body>` создаем контейнер (`<div class="container">`), который ограничивает ширину содержимого и центрирует его на странице. Страница разделена на две основные секции:

1. Секция загрузки файла (`<div id="uploadSection">`): Здесь пользователь может загрузить файл в формате CSV или JSON через элемент `<input type="file">`. Также есть выпадающий список (`<select>`), позволяющий выбрать тип графика (столбчатая, линейная или круговая диаграмма). Кнопка "Создать график" (`<button id="generateChart">`) запускает процесс обработки данных.

2. Секция отображения графика (<div id="chartSection">): В этой секции находится элемент <canvas>, который используется для отрисовки графика с помощью библиотеки Chart.js. Также есть две кнопки: "Вернуться к загрузке" (<button id="backButton">) для возврата к форме загрузки и "Скачать график" (<button id="downloadButton">) для сохранения графика в формате изображения.

Данный код прописываем после тега head:

```
<body>
  <div class="container">
    <div class="card" id="uploadSection">
      <div class="card-header">
        <h1 class="text-center mb-4">Загрузка данных</h1>
      </div>
      <div class="card-body">
        <div class="mb-3">
          <label for="fileUpload" class="form-label">Загрузите
файл с данными</label>
          <input type="file" class="form-control"
id="fileUpload" accept=".csv, .json">
        </div>
        <div class="mb-3">
          <label for="chartType" class="form-label">Выберите
тип графика</label>
          <select class="form-select" id="chartType">
            <option value="bar">Столбчатая
диаграмма</option>
            <option value="line">Линейный график</option>
            <option value="pie">Круговая диаграмма</option>
          </select>
        </div>
        <button class="btn btn-primary"
id="generateChart">Создать график</button>
      </div>
    </div>

    <div class="card hidden" id="chartSection">
      <div class="card-header">
        <h1 class="text-center mb-4">График</h1>
      </div>
      <div class="card-body">
        <div class="chart-container">
          <canvas id="myChart" width="900"
height="600"></canvas>
        </div>
        <div class="button-group">
          <button class="btn btn-secondary"
id="backButton">Вернуться к загрузке</button>
          <button class="btn btn-success"
id="downloadButton">Скачать график</button>
        </div>
      </div>
    </div>
  </div>
```

Для того, чтобы просмотреть файл index.html, необходимо нажать правой кнопкой мыши на файл, и выбрать открыть с помощью, и выбираем

браузер. После того, как прописали код HTML-структуры, на главной странице появится окно загрузки данных (см.рис.1).

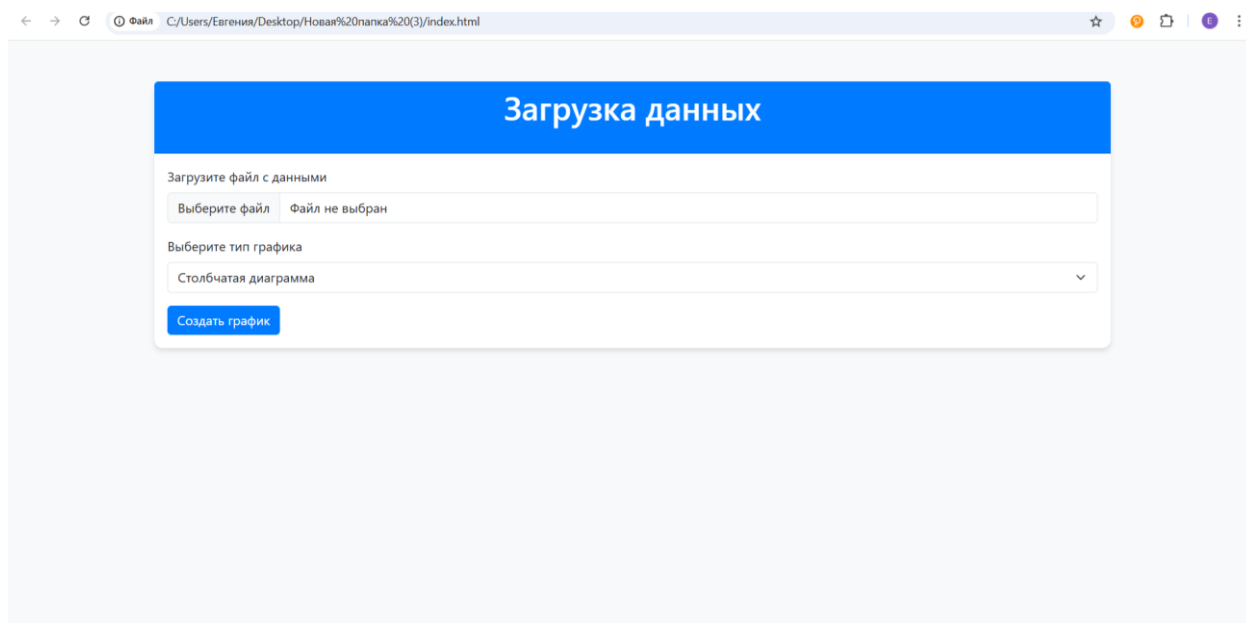


Рисунок 1. Страница загрузки данных

Далее прописываем JavaScript-логику веб приложения. Подключаем библиотеку Chart.js. Библиотека Chart.js подключается через CDN (`<script src="...">`). Она предоставляет инструменты для создания интерактивных графиков и диаграмм на основе элемента `<canvas>`. Следующим прописываем код:

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

Прописываем основной скрипт. Переменная `let myChart`; объявляется для хранения экземпляра графика. Это позволяет управлять графиком в процессе работы приложения. Прописываем код основного скрипта:

```
<script>
    let myChart;
```

Далее прописываем код для обработчика кнопки "Создать график". Когда пользователь нажимает кнопку "Создать график", выполняется следующий процесс:

1. Проверяется, выбран ли файл. Если файл не выбран, выводится предупреждение.
2. Проверяется тип файла. Если файл не является CSV или JSON, выводится сообщение об ошибке.
3. Используется `FileReader` для чтения содержимого файла. В зависимости от типа файла (CSV или JSON) вызывается соответствующая функция для обработки данных (`parseCSV` или `parseJSON`).

4. Если данные успешно обработаны, скрывается секция загрузки файла и отображается секция с графиком. Затем вызывается функция `generateChart`, которая создает график на основе выбранного типа и данных.

Прописываем код после переменной `let myChart`:

```
document.getElementById('generateChart').addEventListener('click', function()
{
    const fileInput = document.getElementById('fileUpload');
    const file = fileInput.files[0];
    if (!file) {
        alert('Пожалуйста, выберите файл для загрузки.');
```

return;

```
    }

    if (file.type !== 'text/csv' && file.type !== 'application/json')
    {
        alert('Неподдерживаемый тип файла. Пожалуйста, загрузите файл
в формате CSV или JSON.');
```

return;

```
    }

    const reader = new FileReader();
    reader.onload = function(e) {
        const rawData = e.target.result;
        const chartType = document.getElementById('chartType').value;

        let data;
        if (file.type === 'text/csv') {
            data = parseCSV(rawData);
        } else if (file.type === 'application/json') {
            data = parseJSON(rawData);
        }

        if (!data || !data.length) {
            alert('Нет данных для отображения.');
```

return;

```
        }

        // Скрываем форму загрузки и показываем график

document.getElementById('uploadSection').classList.add('hidden');

document.getElementById('chartSection').classList.remove('hidden');

        // Генерация графика
        generateChart(chartType, data);
    };
    reader.readAsText(file);
});
```

Далее прописываем код для обработчиков кнопок "Вернуться к загрузке" и "Скачать график". Кнопка "Вернуться к загрузке": При нажатии скрывается секция с графиком и отображается секция загрузки файла. Если график уже был создан, он уничтожается с помощью метода `destroy()`. Кнопка "Скачать график": График преобразуется в изображение в формате PNG с помощью метода `toBase64Image()` (см.рис.3). Затем создается ссылка для

скачивания изображения, которая автоматически активируется. Прописываем код после кода обработчика кнопки "Создать график":

```
// Обработчик кнопки "Вернуться к загрузке"
document.getElementById('backButton').addEventListener('click',
function() {
    // Скрываем график и показываем форму загрузки

document.getElementById('chartSection').classList.add('hidden');

document.getElementById('uploadSection').classList.remove('hidden');

    // Уничтожаем текущий график, если он существует
    if (myChart) {
        myChart.destroy();
    }
});

// Обработчик кнопки "Скачать график"
document.getElementById('downloadButton').addEventListener('click', function()
{
    if (!myChart) {
        alert('График не создан.');
```

Следующим прописываем код функции для обработки данных. Функции для обработки данных:

parseCSV(csv): Эта функция разбирает CSV-файл. Сначала данные разделяются на строки, затем первая строка используется для создания заголовков, а остальные строки преобразуются в объекты. Каждый объект представляет собой строку данных, где ключи — это заголовки, а значения — данные из соответствующих ячеек.

parseJSON(json): Эта функция разбирает JSON-файл. Данные преобразуются в массив объектов, где каждый объект содержит вопрос и количество ответов "Да" и "Нет".

Прописываем код после обработчика кнопки "Скачать график":

```
function parseCSV(csv) {
    const lines = csv.split('\n');
    const headers = lines[0].split(';');
    const data = [];
```

```
for (let i = 1; i < lines.length; i++) {
  const obj = {};
  const currentLine = lines[i].split(';');
  for (let j = 0; j < headers.length; j++) {
    obj[headers[j].trim()] = currentLine[j].trim();
  }
  data.push(obj);
}
return data;
}

function parseJSON(json) {
  const rawData = JSON.parse(json);
  const data = [];
  for (const [question, answers] of Object.entries(rawData)) {
    data.push({
      Вопрос: question,
      Да: answers.да,
      Нет: answers.нет
    });
  }
  return data;
}
```

Далее, чтобы был функционал для создания графика, необходимо добавить функцию `generateChart(type, data)`, которая создает график на основе выбранного типа и данных:

1. Получает контекст элемента `<canvas>`.
2. Если график уже существует, он уничтожается.
3. Подготавливает данные для графика: извлекает вопросы, ответы "Да" и "Нет".
4. В зависимости от типа графика (столбчатая, линейная или круговая диаграмма) формирует данные для отрисовки.
5. Создает конфигурацию для графика, включая настройки осей, легенды и других параметров.
6. Инициализирует график с помощью библиотеки `Chart.js`.

Прописываем код после функции для обработки данных:

```
function generateChart(type, data) {
  const ctx =
document.getElementById('myChart').getContext('2d');
  if (myChart) {
    myChart.destroy();
  }

  const labels = data.map(item => item.Вопрос);
  const yesData = data.map(item => parseInt(item.Да));
  const noData = data.map(item => parseInt(item.Нет));

  let chartData;
  if (type === 'pie') {
    // Для круговой диаграммы объединяем данные "Да" и "Нет"
    chartData = {
      labels: labels.map((label, index) => `${label} (Да:
${yesData[index]}, Нет: ${noData[index]})`),
      datasets: [{
```

```

        label: 'Ответы',
        data: yesData.concat(noData), // Объединяем
        backgroundColor: generateColors(labels.length *
данные "Да" и "Нет"
2), // Генерация цветов для каждого вопроса
        borderColor: 'rgba(255, 255, 255, 1)',
        borderWidth: 1
    }
  ]
};
} else {
  // Для других типов графиков используем два набора
данных
  chartData = {
    labels: labels,
    datasets: [
      {
        label: 'Да',
        data: yesData,
        backgroundColor: 'rgba(75, 192, 192, 0.2)',
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 1
      },
      {
        label: 'Нет',
        data: noData,
        backgroundColor: 'rgba(255, 99, 132, 0.2)',
        borderColor: 'rgba(255, 99, 132, 1)',
        borderWidth: 1
      }
    ]
  };
}

const config = {
  type: type,
  data: chartData,
  options: {
    responsive: true,
    maintainAspectRatio: false,
    scales: {
      y: {
        beginAtZero: true
      },
      x: {
        ticks: {
          autoSkip: false,
          maxRotation: 90,
          minRotation: 90
        }
      }
    },
    plugins: {
      legend: {
        position: 'top',
      }
    }
  }
};
myChart = new Chart(ctx, config);
}

```

Создаем заключительную функцию `generateColors(count)`. Функция генерирует массив случайных цветов в формате RGBA. Это используется для

окрашивания секторов круговой диаграммы или столбцов на графике. Каждый цвет создается случайным образом, что делает график более визуально привлекательным.

```
function generateColors(count) {
    const colors = [];
    for (let i = 0; i < count; i++) {
        const color = `rgba(${Math.floor(Math.random() * 255)},
${Math.floor(Math.random() * 255)}, ${Math.floor(Math.random() * 255)},
0.6)`;
        colors.push(color);
    }
    return colors;
}
</script>
</body>
</html>
```

Далее проверяем работоспособность веб приложения. Обновляем страницу в браузере. При добавлении файла результата опроса и выборе необходимого графика переходим на страницу графика (см.рис.2). Также проверяем работу кнопки «Скачать график», если кнопка работает, то скачивается график в формате png (см.рис.3).

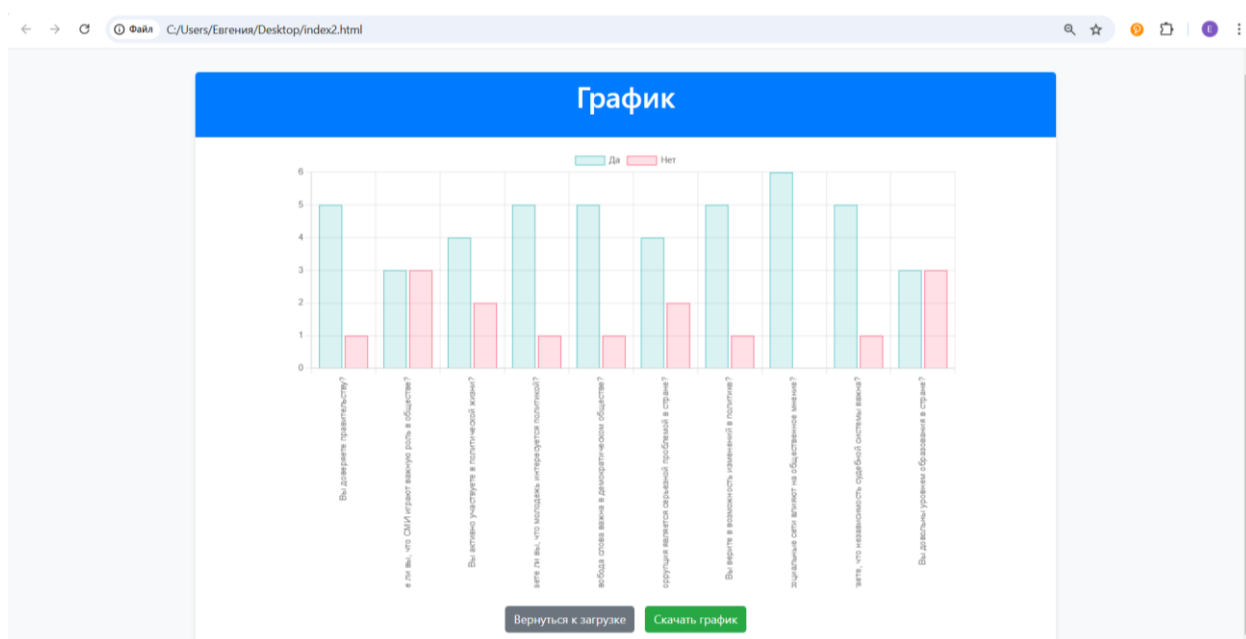


Рисунок 2. График

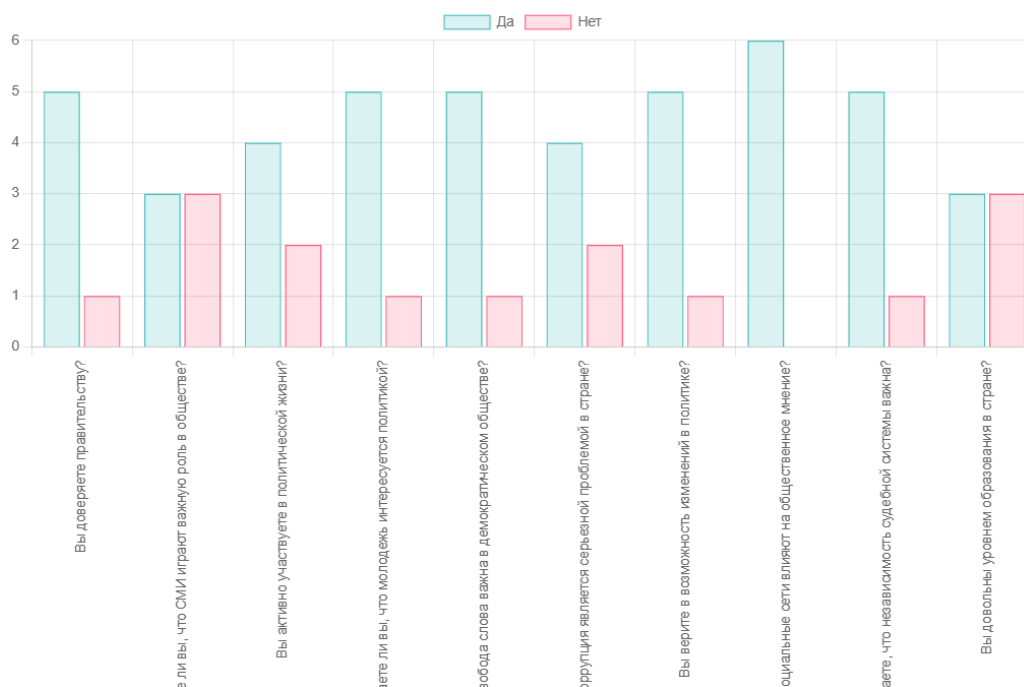


Рисунок 3. Скачанный график

4 Выводы

В данной работе был представлен процесс разработки веб-приложения для анализа и визуализации данных с использованием языка программирования JavaScript, библиотеки Chart.js, а также языков HTML и CSS для создания пользовательского интерфейса. Приложение позволяет пользователям загружать данные в форматах CSV и JSON, выбирать тип графика (столбчатая, линейная или круговая диаграмма) и визуализировать информацию в интерактивном формате. Кроме того, реализована возможность экспорта созданных графиков в виде изображений, что делает приложение удобным инструментом для анализа и презентации данных. Научная статья может служить методическим пособием в учебной деятельности.

Библиографический список

1. Данилин Д. А., Зиновьев Я. В., Кузьмин К. М. Анализ технологий веб-программирования для создания модулей визуализации и загрузки данных информационных систем // Вестник Пензенского государственного университета. 2019. №. 3 (27). С. 76-79.
2. Бизяркин Я. С. Визуализация данных на JAVASCRIPT: сравнение apі и производительности популярных библиотек // Научно-технические инновации и веб-технологии. 2022. № 1. С. 24-32.
3. Иванов О. А., Вдовенко В. В. Визуализация большого количества данных в клиентском приложении //Актуальные проблемы авиации и космонавтики. 2011. Т. 1. №. 7. С. 358-359.
4. Ломаков А. В. Современные технологии визуализации данных // Современные технологии в науке и образовании - СТНО-2020: Сборник трудов III Международного научно-технического форума: в 10 т., Рязань,

- 04–06 марта 2020 года. Т. 5. Рязань: Рязанский государственный радиотехнический университет, Издательство «BookJet», 2020. С. 156-162.
5. Рябинин К. В. Разработка адаптивного мультиплатформенного визуализатора результатов научных расчётов для высокопроизводительных вычислительных систем // XXII Международная конференция по компьютерной графике и машинному зрению «ГрафиКон. 2012. С. 193-198.