Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Нижегородский государственный технический

университет им. Р.Е.Алексеева

Институт радиоэлектроники и информационных технологий

Кафедра прикладной математики и информатики

Отчёт № 7

«HTTP запросы, AJAX»

по дисциплине

web-разработка

Выполнил:

студент группы 23-ПМ-1

Блинов А. С.

Проверил:

Курушин Е. А.

Нижний Новгород

2025 год

Оглавление

Цели и задачи

В ходе данной работы были поставлены следующая цель:

Изучить методы отправки и обработки HTTP-запросов в JavaScript, включая классический подход с XMLHttpRequest и современный способ с использованием fetch().

Для из реализации нужно выполнить следующие задачи:

1. Выполните GET запрос объектом XMLHttpRequest. Обработайте полученные данные и выведите их на страницу в соответствующий элемент.

2. Выполните POST запрос методом fetch(). Обработайте полученные данные (например, добавление на страницу, вывод в консоль, alert и прочее).

Теоретическая справка

Современные веб-приложения активно взаимодействуют с серверами, запрашивая и отправляя данные без перезагрузки страницы. Это обеспечивается технологиями асинхронного обмена данными, такими как AJAX, Fetch API и XMLHttpRequest. В этом разделе рассматриваются ключевые понятия, связанные с HTTP-запросами, их структурой и выполнением в JavaScript.

1. Что такое AJAX?

AJAX (Asynchronous JavaScript and XML) – это подход к созданию интерактивных веб-приложений, при котором данные обмениваются с сервером в фоновом режиме без перезагрузки страницы.

Принцип работы: Браузер отправляет HTTP-запрос в фоне, получает ответ (обычно в JSON или XML) и динамически обновляет интерфейс.

Примеры использования: Автодополнение поиска, загрузка новых сообщений в чате, формы входа без перезагрузки.

Технологии: Основан на XMLHttpRequest или современном fetch().

2. Что такое GET и POST запросы? Чем они отличаются?

GET и POST – основные методы HTTP-запросов, отличающиеся способом передачи данных. Отличия представлены в таблице 1.

Характеристика	**GET**	**POST**
Назначение	Получение данных	Отправка данных
Данные в URL	Передаются в строке запроса	Передаются в теле запроса
Кэширование	Может кэшироваться	Не кэшируется

Безопасность	Менее безопасен (данные в URL)	Безопаснее (данные скрыты)
Ограничение длины	Ограничен длиной URL (~2048 симв.)	Нет жестких ограничений

Таблица 1 – Отличия GET и POST

Примеры:

GET – загрузка списка товаров, поиск.

POST – отправка формы входа, загрузка файла.

3. Что такие заголовки запроса и какие они могут быть?

Заголовки HTTP-запроса (Request Headers) – это метаданные, которые передаются клиентом на сервер для описания запроса.

Основные заголовки:

- Content-Type – тип передаваемых данных (application/json, text/html).

- Authorization – данные аутентификации (токены, Basic Auth).

- Accept – допустимые форматы ответа (application/json).

- User-Agent – информация о браузере и ОС.

- Cache-Control – управление кэшированием (no-cache).

Пример в JavaScript (fetch):

```
fetch('https://api.example.com/data', {
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer token123'
  }
```

});

4. Что такое заголовки ответа и какие они могут быть?

Заголовки HTTP-ответа (Response Headers) – метаданные, которые сервер отправляет клиенту вместе с данными.

Основные заголовки:

- Content-Type – тип возвращаемых данных (application/json).

- Status – код состояния (200, 404, 500).

- Set-Cookie – установка cookies.

- Access-Control-Allow-Origin – политика CORS.

- Cache-Control – настройки кэширования.

Пример проверки в JavaScript:

```
fetch('https://api.example.com/data')
  .then(response => {
    console.log(response.headers.get('Content-Type'));
  });
```

5. Как выполнить HTTP-запрос?

В JavaScript есть два основных способа:

- Через XMLHttpRequest (устаревший, но поддерживается везде):

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.example.com/data');
xhr.onload = () => console.log(xhr.responseText);
xhr.send();
```

- Через fetch() (современный способ):

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

6. Что такое XMLHttpRequest и fetch и в чем их отличие?

XMLHttpRequest (XHR) — старый, но до сих пор поддерживаемый способ выполнения AJAX-запросов.

fetch() — современный, более удобный и читаемый способ взаимодействия с сервером. Возвращает промисы и легче интегрируется с асинхронным кодом.

Отличия представлены в таблице 2.

Критерий	**XMLHttpRequest**	**fetch**
Синтаксис	Громоздкий, на событиях	Чище, на Promise
Поддержка JSON	Требует ручного парсинга	Встроенный .json()
Отмена запроса	Через xhr.abort()	Через AbortController
CORS	Обрабатывает непрозрачно	Более строгая обработка
Поддержка	Все браузеры	Не поддерживает IE11

Таблица 2 – Отличия XMLHttpRequest и fetch

Вывод: fetch() удобнее и современнее, но XMLHttpRequest может быть полезен для старых браузеров.

7. Что такое CORS?

CORS (Cross-Origin Resource Sharing) – механизм, разрешающий или запрещающий запросы между разными доменами (например, site.com → api.site.com).

Как работает? Браузер отправляет предварительный запрос (OPTIONS), проверяя заголовки (Access-Control-Allow-Origin).

Ошибка CORS: Возникает, если сервер не разрешает跨-origin запросы.

Решение:

- Настроить сервер (Access-Control-Allow-Origin: *).

- Использовать прокси.

- Для тестов – отключить CORS в браузере (не для продакшена!).

Пример правильного заголовка:

Access-Control-Allow-Origin: https://your-site.com

Реализация

В ходе выполнения работы были освоены два подхода к взаимодействию с веб-API: GET-запрос через классический XMLHttpRequest и POST-запрос с применением современного Fetch API. Оба метода продемонстрировали работу с внешними сервисами и динамическое обновление содержимого веб-страницы.

1. Получение данных через XMLHttpRequest

Для демонстрации GET-запроса использовался API GitHub, предоставляющий информацию о пользователях. Реализация включает:

- Инициализацию объекта XMLHttpRequest

- Настройку запроса к эндпоинту /users/octocat

- Установку заголовка Content-Type: application/json

- Обработку ответа с преобразованием JSON-данных

При успешном выполнении (статус 200) извлекаются:

- Имя пользователя

- Логин

- Биография

- Аватар (с визуальным оформлением)

Элементы динамически создаются и добавляются в контейнер data-container. В случае ошибки выполняется её логирование в консоль.

2. Отправка данных через Fetch API

POST-запрос реализован к тестовому API reqres.in, имитирующему создание пользователя. Особенности реализации:

- Формирование объекта с данными (имя и должность)

- Настройка запроса с указанием:

- Метода POST

- Заголовков Content-Type

- Тела запроса в JSON-формате

- Цепочка обработки ответа:

- Преобразование из JSON

- Отображение результата (имя, должность, ID и временная метка)

- Логирование ошибок при необходимости

Результаты выполнения:

- GET-запрос успешно получает и отображает профиль пользователя GitHub

- POST-запрос корректно создаёт новую запись и выводит ответ сервера

- Оба метода демонстрируют различные подходы к работе с API

- Реализована обработка как успешных, так и ошибочных сценариев

На рисунке 1 представлена страница.

Рисунок 1 – Страница

Вывод

В рамках лабораторного исследования были изучены основные принципы обмена данными между клиентом и сервером с использованием JavaScript. В ходе работы применены два способа отправки HTTP-запросов:

- Через классический интерфейс XMLHttpRequest для выполнения GET-запроса с последующей обработкой и отображением полученных данных на странице.

- С применением современного API fetch() для отправки POST-запроса, включающего передачу данных на сервер и вывод ответа в консоли, а также в виде пользовательского оповещения.

Дополнительно отработаны ключевые аспекты работы с JSON-форматом, управления асинхронными запросами и динамического обновления интерфейса на основе полученных данных. В результате поставленные задачи выполнены, что позволило закрепить навыки, необходимые для организации взаимодействия фронтенд-части веб-приложений с серверной стороной.

Приложение

```
function getRequest() {
    const xhr = new XMLHttpRequest();

    xhr.open('GET', 'https://api.github.com/users/octocat', true); // GitHub API
    xhr.setRequestHeader('Content-Type', 'application/json');

    xhr.onload = function () {
        if (xhr.status === 200) {
            const data = JSON.parse(xhr.responseText);

            const dataContainer = document.getElementById('data-container');
            dataContainer.innerHTML = ''; // Очищаем старое

            // Создаём и вставляем элементы
            const nameElement = document.createElement('p');
            nameElement.textContent = `Имя: ${data.name}`;

            const loginElement = document.createElement('p');
            loginElement.textContent = `Логин: ${data.login}`;

            const bioElement = document.createElement('p');
            bioElement.textContent = `Описание: ${data.bio}`;

            const avatar = document.createElement('img');
            avatar.src = data.avatar_url;
            avatar.alt = 'Avatar';
            avatar.style.width = '150px';
            avatar.style.borderRadius = '10px';

            dataContainer.appendChild(nameElement);
            dataContainer.appendChild(loginElement);
            dataContainer.appendChild(bioElement);
            dataContainer.appendChild(avatar);

        } else {
            console.error('Ошибка при выполнении GET запроса');
        }
    };

    xhr.send();
}

getRequest();


// 2. Выполняем POST запрос с использованием fetch()
function postRequest() {
    const postData = {
        name: 'Alice',
        job: 'Engineer'
    };

    // Отправляем POST запрос
    fetch('https://reqres.in/api/users', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(postData)
    })
        .then(response => response.json())
        .then(data => {
```

```
            console.log('Ответ от сервера:', data);

            // Выводим результат на страницу
            const container = document.getElementById('data-container');
            const result = document.createElement('p');
            result.textContent = `Создан пользователь: ${data.name}, Должность: ${data.job}, ID: ${data.id}, Время:
${data.createdAt}`;
            container.appendChild(result);
        })
        .catch(error => {
            console.error('Ошибка при выполнении POST запроса:', error);
        });
}

// Запускаем
postRequest();
```