**Лабораторная работа №26**

**Тема работы:** Разработка программ с применением fetch запросов.

**Цель работы:** закрепить навыки по работе с fetch запросами.

**Теоретические сведения**

**Fetch**

JavaScript может отправлять сетевые запросы на сервер и подгружать новую информацию по мере необходимости.

Например, мы можем использовать сетевой запрос, чтобы:

* Отправить заказ,
* Загрузить информацию о пользователе,
* Запросить последние обновления с сервера,
* …и т.п.

Для сетевых запросов из JavaScript есть широко известный термин «AJAX» (аббревиатура от **A**synchronous **J**avaScript **A**nd **X**ML). XML мы использовать не обязаны, просто термин старый, поэтому в нём есть это слово. Возможно, вы его уже где-то слышали.

Есть несколько способов делать сетевые запросы и получать информацию с сервера.

Метод fetch() — современный и очень мощный, поэтому начнём с него. Он не поддерживается старыми (можно использовать полифил), но поддерживается всеми современными браузерами.

Базовый синтаксис:

let promise = fetch(url, [options])

* **url** – URL для отправки запроса.
* **options** – дополнительные параметры: метод, заголовки и так далее.

Без options это простой GET-запрос, скачивающий содержимое по адресу url.

Браузер сразу же начинает запрос и возвращает промис, который внешний код использует для получения результата.

Процесс получения ответа обычно происходит в два этапа.

**Во-первых, promise выполняется с объектом встроенного класса**[**Response**](https://fetch.spec.whatwg.org/#response-class)**в качестве результата, как только сервер пришлёт заголовки ответа.**

На этом этапе мы можем проверить статус HTTP-запроса и определить, выполнился ли он успешно, а также посмотреть заголовки, но пока без тела ответа.

Промис завершается с ошибкой, если fetch не смог выполнить HTTP-запрос, например при ошибке сети или если нет такого сайта. HTTP-статусы 404 и 500 не являются ошибкой.

Мы можем увидеть HTTP-статус в свойствах ответа:

* **status** – код статуса HTTP-запроса, например 200.
* **ok** – логическое значение: будет true, если код HTTP-статуса в диапазоне 200-299.

Например:

let response = await fetch(url);

if (response.ok) { // если HTTP-статус в диапазоне 200-299

// получаем тело ответа (см. про этот метод ниже)

let json = await response.json();

} else {

alert("Ошибка HTTP: " + response.status);

}

**Во-вторых, для получения тела ответа нам нужно использовать дополнительный вызов метода.**

Response предоставляет несколько методов, основанных на промисах, для доступа к телу ответа в различных форматах:

* **response.text()** – читает ответ и возвращает как обычный текст,
* **response.json()** – декодирует ответ в формате JSON,
* **response.formData()** – возвращает ответ как объект FormData (разберём его в [следующей главе](https://learn.javascript.ru/formdata)),
* **response.blob()** – возвращает объект как [Blob](https://learn.javascript.ru/blob) (бинарные данные с типом),
* **response.arrayBuffer()** – возвращает ответ как [ArrayBuffer](https://learn.javascript.ru/arraybuffer-binary-arrays) (низкоуровневое представление бинарных данных),
* помимо этого, response.body – это объект [ReadableStream](https://streams.spec.whatwg.org/#rs-class), с помощью которого можно считывать тело запроса по частям. Мы рассмотрим и такой пример несколько позже.

Например, получим JSON-объект с последними коммитами из репозитория на GitHub:

let url = 'https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits';

let response = await fetch(url);

let commits = await response.json(); // читаем ответ в формате JSON

alert(commits[0].author.login);

То же самое без await, с использованием промисов:

fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits')

.then(response => response.json())

.then(commits => alert(commits[0].author.login));

Для получения ответа в виде текста используем await response.text() вместо .json():

let response = await fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits');

let text = await response.text(); // прочитать тело ответа как текст

alert(text.slice(0, 80) + '...');

В качестве примера работы с бинарными данными, давайте запросим и выведем на экран логотип [спецификации «fetch»](https://fetch.spec.whatwg.org/) (см. главу [Blob](https://learn.javascript.ru/blob), чтобы узнать про операции с Blob):

let response = await fetch('/article/fetch/logo-fetch.svg');

let blob = await response.blob(); // скачиваем как Blob-объект

// создаём <img>

let img = document.createElement('img');

img.style = 'position:fixed;top:10px;left:10px;width:100px';

document.body.append(img);

// выводим на экран

img.src = URL.createObjectURL(blob);

setTimeout(() => { // прячем через три секунды

img.remove();

URL.revokeObjectURL(img.src);

}, 3000);

**Важно:**

Мы можем выбрать только один метод чтения ответа.

Если мы уже получили ответ с response.text(), тогда response.json() не сработает, так как данные уже были обработаны.

let text = await response.text(); // тело ответа обработано

let parsed = await response.json(); // ошибка (данные уже были обработаны)

**[Заголовки ответа](https://learn.javascript.ru/fetch" \l "zagolovki-otveta)**

Заголовки ответа хранятся в похожем на Map объекте response.headers.

Это не совсем Map, но мы можем использовать такие же методы, как с Map, чтобы получить заголовок по его имени или перебрать заголовки в цикле:

let response = await fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits');

// получить один заголовок

alert(response.headers.get('Content-Type')); // application/json; charset=utf-8

// перебрать все заголовки

for (let [key, value] of response.headers) {

alert(`${key} = ${value}`);

}

**[Заголовки запроса](https://learn.javascript.ru/fetch" \l "zagolovki-zaprosa)**

Для установки заголовка запроса в fetch мы можем использовать опцию headers. Она содержит объект с исходящими заголовками, например:

let response = fetch(protectedUrl, {

headers: {

Authentication: 'secret'

}

});

Есть список [запрещённых HTTP-заголовков](https://fetch.spec.whatwg.org/#forbidden-header-name), которые мы не можем установить:

* Accept-Charset, Accept-Encoding
* Access-Control-Request-Headers
* Access-Control-Request-Method
* Connection
* Content-Length
* Cookie, Cookie2
* Date
* DNT
* Expect
* Host
* Keep-Alive
* Origin
* Referer
* TE
* Trailer
* Transfer-Encoding
* Upgrade
* Via
* Proxy-\*
* Sec-\*

Эти заголовки обеспечивают достоверность данных и корректную работу протокола HTTP, поэтому они контролируются исключительно браузером.

**[POST-запросы](https://learn.javascript.ru/fetch" \l "post-zaprosy)**

Для отправки POST-запроса или запроса с другим методом, нам необходимо использовать fetch параметры:

* **method** – HTTP метод, например POST,
* **body** – тело запроса, одно из списка:
  + строка (например, в формате JSON),
  + объект FormData для отправки данных как form/multipart,
  + Blob/BufferSource для отправки бинарных данных,
  + [URLSearchParams](https://learn.javascript.ru/url) для отправки данных в кодировке x-www-form-urlencoded, используется редко.

Чаще всего используется JSON.

Например, этот код отправляет объект user как JSON:

let user = {

name: 'John',

surname: 'Smith'

};

let response = await fetch('/article/fetch/post/user', {

method: 'POST',

headers: {

'Content-Type': 'application/json;charset=utf-8'

},

body: JSON.stringify(user)

});

let result = await response.json();

alert(result.message);

Заметим, что так как тело запроса body – строка, то заголовок Content-Type по умолчанию будет text/plain;charset=UTF-8.

Но, так как мы посылаем JSON, то используем параметр headers для отправки вместо этого application/json, правильный Content-Type для JSON.

**Порядок выполнения работы**

Задача 1

Реализовать на странице поле ввода текста и кнопку, по нажатию на которую будет добавляться на странице таблица с данными о погоде города, введенного в поле, а также рекомендации относительно данных о погоде. В данные о погоде должны входить: температура воздуха, влажность, осадки, направление и скорость ветра, иконка погоды.

Ссылка на документацию по API: <https://openweathermap.org/current>, <https://openweathermap.org/weather-conditions>

Ключ для API - 8643e5fa4d67cb1ad3c160e1d6c66d90