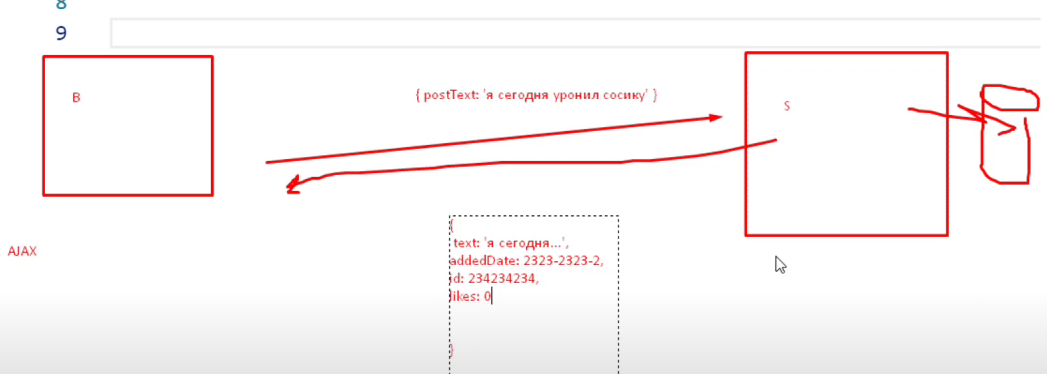
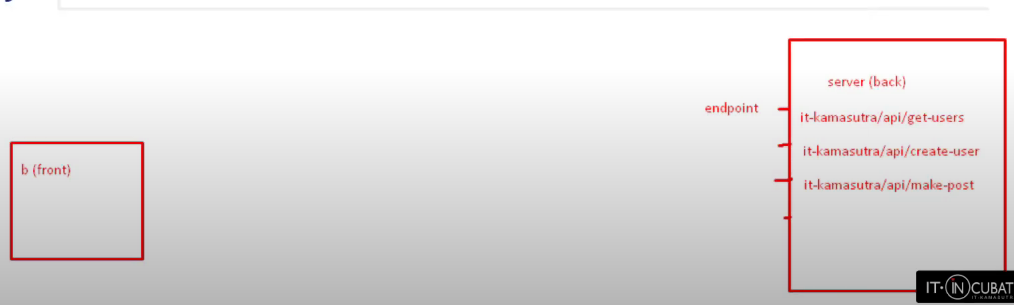
# Введение в ajax, jQuery, get-запрос

AJAX – это асинхронные запросы, т.е. без перезагрузки страницы применяются изменения на клиентской части. Каждая перезагрузка страницы, это отправка запроса на сервер.



В данном примере хотим добавить пост на страницу с текстом “Я сегодня….”. Данная строка отправляется на сервер. Он в свою очередь записывает эту строку в БД и возвращает данные в json формате необходимые для отрисовки нового поста на странице.

Для тренировки запроса: <https://jsonplaceholder.typicode.com/>



Информация от backend разработчика.



Endpoint – это точки по которым можно обратиться на сервер для получения той ли иной информации. Их прописывают backend разработчики.

# JavaScript, success, callback, обработка ответа

Для обработки запроса нужен callback. Это когда мы отдаем функцию кому-то и говорим выполнить эту функцию, когда придет время.

Callback зачастую используется, когда речь идет об асинхронном коде. Т.е., например запрос, мы не знаем когда он завершиться. Но по завершению функция выполняется.

Асинхронный код – это выполнение кода строка за строкой.

Пример асинхронности №1

console.log(0);

setTimeout(*function*() {

    console.log(1);

}, 1000);

console.log(2);

Результат



Пример асинхронности №2

console.log(0);

$.ajax('https://jsonplaceholder.typicode.com/users/1', {

    success: *function*(*data*) {

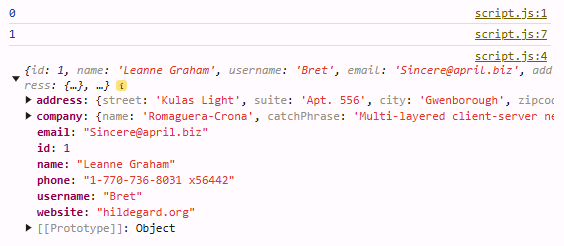
        console.log(*data*);

    }

});

console.log(1);

Результат



Это происходит потому, что ответ на запрос происходит не сразу, а через какое-то время.

С полученными данными можем проводить различные манипуляции

$.ajax('https://jsonplaceholder.typicode.com/users', {

    success: *function*(*data*) {

*data*.forEach(*element* *=>* {

*let* name = document.createElement('div');

            name.innerHTML = *element*.name;

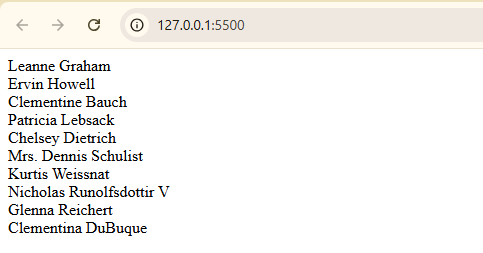
            document.querySelector('body').append(name);

        });

    }

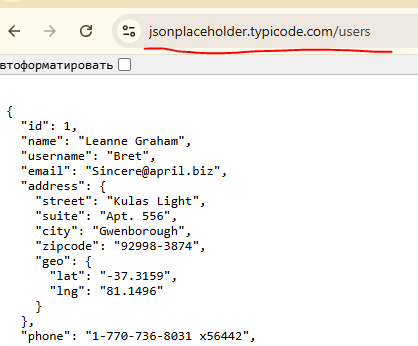
});

Результат

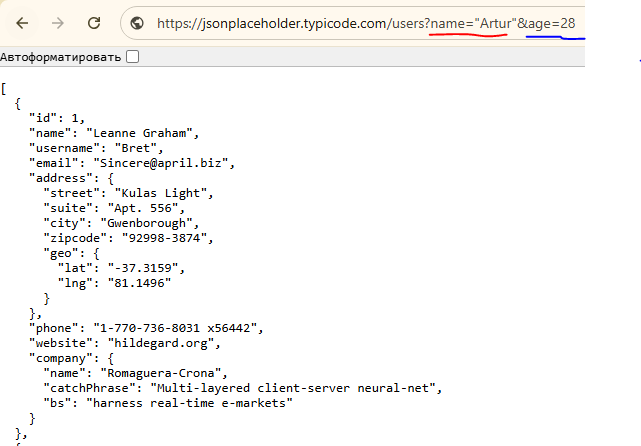


# get, query params, get-параметры, query string

Используем этот тип запроса если мы хотим от сервера получать данные. Простейший get запрос, это обычный ввод данных в url.



Если мы хотим передать данные на сервер при помощи метода get, то делать это можно при помощи get-параметров после знака “?”.



В данном случае запрашиваем у сервера данные через endpoint и одновременно отправляем данные двумя get-параметрами (соединяем их знаком &). Эти параметры указываются для того, чтобы получить от сервера не все данные, а именно такие, которые будут удовлетворять тем условиям, которые указаны в get-параметрах. Метод запроса и параметры, которые можем отправлять определяет серверный разработчик!!!

Но у строки есть ограничение по длине. Поэтому нельзя передать большое количество данных.

*const* resultBlock = document.querySelector("#result");

*const* clickMeButton = document.querySelector("#click-me");

clickMeButton.addEventListener("click", makeRequest);

*function* makeRequest() {

    $.ajax('https://jsonplaceholder.typicode.com/users', {

        success: *function* (*data*) {

*data*.forEach(*element* *=>* {

*let* name = document.createElement('div');

                name.innerHTML = *element*.name;

                document.querySelector('#result').append(name);

            });

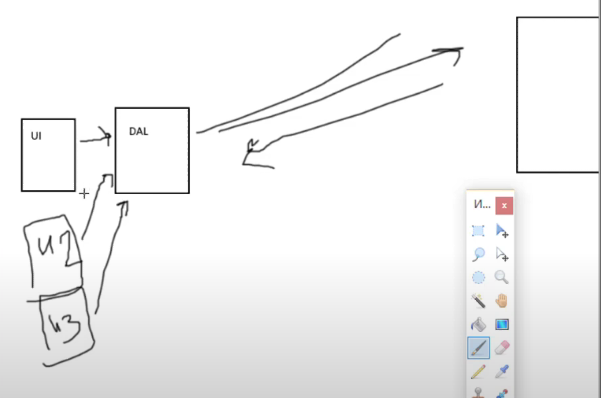
        }

    });

}

# Ui-dal (data access layer)

Разбиение программы на два уровня:



UI – получает данные со страницы (элементы)

Dal (уровень доступа к данным) – получает данные из UI через callback и взаимодействует с сервером.

В нужный момент UI пинает DAL, чтобы тот сделал запрос на сервер. В dal не должно быть не единой строки кода, которая работала бы с html.

Script.js (ui)

*const* resultBlock = document.querySelector("#result");

*const* clickMeButton = document.querySelector("#click-me");

*const* number = document.querySelector("#number");

clickMeButton.addEventListener("click", () *=>* {

    makeRequest(number.value, onDataReceived);

});

*function* onDataReceived(*data*) {

*data*.forEach(*element* *=>* {

*let* name = document.createElement('div');

        name.innerHTML = *element*.name;

        document.querySelector('#result').append(name);

    });

}

Service.js (dal)

*function* makeRequest(*number*, *onDataReceived*) {

    $.ajax(`https://jsonplaceholder.typicode.com/comments?postId=${*number*}`, {

        success: *onDataReceived*

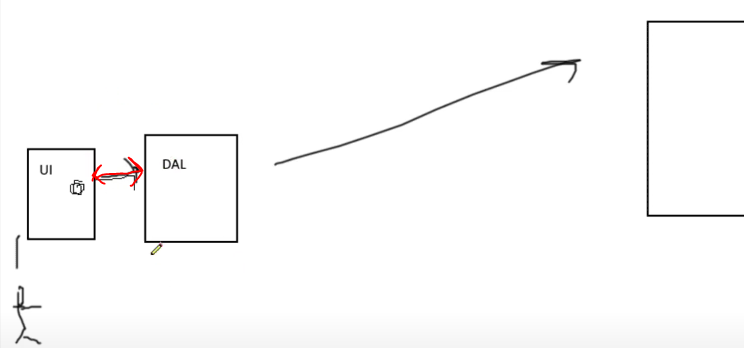
    });

}

# Promise, axios

Метод ajax запускает асинхронную операцию и возвращает промис.

Promise – это обещание, которое говорит о том, что когда-то асинхронная операция закончится и через промис можно будет к этому результату достучаться.



Script.js

*const* resultBlock = document.querySelector("#result");

*const* clickMeButton = document.querySelector("#click-me");

*const* number = document.querySelector("#number");

clickMeButton.addEventListener("click", () *=>* {

    //Получаем промис

*const* promise = makeRequest(number.value);

    //В случае успеха ЗАТЕМ нужно вызвать callback

    promise

        .then(onDataReceived);

});

*function* onDataReceived(*data*) {

*data*.forEach(*element* *=>* {

*let* name = document.createElement('div');

        name.innerHTML = *element*.name;

        document.querySelector('#result').append(name);

    });

}

Servise.js

//Promise ajax

*function* makeRequest(*number*) {

    //ajax вернул промис

*const* promise = $.ajax(`https://jsonplaceholder.typicode.com/comments?postId=${*number*}`);

    //функция возвращает промис

    return promise;

}

Библиотека axios для запросов. Применяется в react.

<https://github.com/axios/axios>

<https://www.youtube.com/watch?v=fDVhl3DGkD0&t=131s> – курс

Просто меняем метод в service.js и все должно работать. Так происходит потому, что мы выполняли разделение на UI и DAL.

*function* makeRequest(*number*) {

*const* promise = axios.get(`https://jsonplaceholder.typicode.com/comments?postId=${*number*}`);

    //функция возвращает промис

    return promise.then((*response*) *=>* {

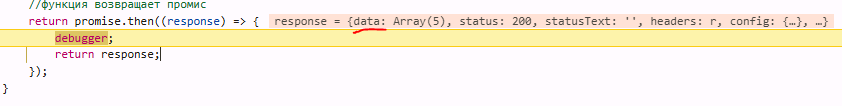
        return *response*.data;

    });

}

Отличие ajax от axios заключается в том, что в ajax в качестве ответа приходил массив объектов, а в axios приходит большой объект, в котором нам нужно обращаться к свойству data (массив).

Ответ axios (response)



Ответ ajax



# Request payload, POST, отправка данных на сервер

Отличие get от post заключается в том, что при post запросе можно отправлять большое количество данных в body на сервер.

Возможно настройки сервера на jsonplaceholder при post запросе нет возврата данных. Есть только при get.

Идемпотентность (get запрос) - это обеспечение возможности многократного вызова запроса с гарантией того, что состояние системы изменится только один раз. Смысл её в том, что наш запрос никак не меняет состояние сервера.

Post запросы меняют состояние системы.