# Intro

function User(name){

this.name = name;

this.sayHello = () => {

console.log('Hello ', this.name);

};

}

var Jhon = new User('Jhon');

Jhon.sayHello();

PS C:\Users\24Scroll\Desktop\Nodejs\1\Samples\001\_Simple> node main.js

## Module

require('./mod');

console.log('Hello from MAIN module!');

## Global

//Переменные обьявленные внутри модуля, являются локальными для модуля

var x = 10;

function test(){

console.log('Test function, number = ', x);

};

// что бы переменная была доступна в другом модуле, можно сделать ее свойством глобального обьекта

// для доступа к глобальному объекту используется переменная global

global.number = x;

global.func = test;

require('./mod');

console.log(number);

func();

## Exports

//На практике обьект global не используется

//Переменные обьявленные внутри модуля, являются локальными для модуля

var x = 10;

function test(){

console.log('Test function, number = ', x);

};

// для того чтобы сделать переменные доступны в другом модуле, необходимо добавить их в обьект exports

// exports - объект, который возвращается функцией require

exports.number = x;

exports.func = test;

// exports = x; // Error

var obj = require('./mod');

console.log(obj);

## Module

console.log('test module');

module.exports.x = 10;

module.exports.y = 15;

var mod = require('./mod.js');

// Объект module хранит информацию о текущем модуле

console.log(module);

//Module {

// id: '.', - путь к запускаемому файлу

// exports: {}, - обьект который возвращается функцией require

// parent: null, - родительский модуль

// filename: 'D:\\Node\\006\_Module\\main.js', - Абсолютный путь к файлу

// loaded: false, - статус обработки модуля

// children: - дочерние модули

// [ Module {

// id: 'D:\\Node\\006\_Module\\mod.js',

// exports: {},

// parent: [Circular],

// filename: 'D:\\Node\\006\_Module\\mod.js',

// loaded: true,

// children: [],

// paths: [Object] } ],

// paths: - пути по которым происходит поиск модуля

// [ 'D:\\Node\\006\_Module\\node\_modules',

// 'D:\\Node\\node\_modules',

// 'D:\\node\_modules' ] }

## Connect module

function CreateConnection(){

this.connect = () =>{

console.log('Connection established!');

}

}

function testConnection(){

console.log('Test connection...');

new CreateConnection().connect();

}

// проверяем является ли модуль подключаемым или запускаемым

// если модуль подключается, мы возвращаем функцию, если модуль запускается, устанавливаем тестовое соединение

if(module.parent){

module.exports = CreateConnection;

}

else{

testConnection();

}

var db = require('./dbConnection');

var cn = new db();

cn.connect();

## Load modules

/\* Алгоритм поиска модулей при вызыове функции require('имя\_модуля')

1. Если модуль является системным, происходит его загрузка

2. Если 'имя\_модуля' начинается со знака '/' - поиск модуля будет производится в корне файловой системы

3. Если 'имя\_модуля' начинается со знака '/', './', '../' - поиск осуществляется несколькими способами:

3.1 Происходит поиск файла 'имя\_модуля' по указанному пути (см. Загрузка файла)

3.2 Происходит поиск директории имя которой 'имя\_модуля' по указанному пути (см. Загрузка директории

4. Поиск папки node\_modules

4.1 Поиск модуля происходит в директории node\_modules

4.2 Если в текущем каталоге нет каталога node\_modules, происходит переход в каталог уровнем выше, и производится поиск в нее

5. Ошибка, модуль не найден

\*/

// Загрузка файла

// 1. Если 'имя\_модуля' или 'имя\_модуля.js' происходит загрузка JavaScript файла

// 2. Если 'имя\_модуля.json', происходит парсинг файла и загружается JavaScript объект

// 3. Если 'имя\_модуля.node', происходит загрузка бинарного файла

// Загрузка директории

// 1. Если 'имя\_модуля' соответствует имени директории и в ней есть файл package.json, происходит парсинг сожержимого в объект

// и поиск свойства main (в свойстве находится путь к загружаемым модулям).

// Если все прошло успешно, со свойства извлекается значение и происходит загрузка файла (см. Загрузка файла)

// 2. Если предыдущий пункт не выполнен, происходит загрузка файла с именем index (см. Зашрузка индекс файла)

// Загрузка index файла

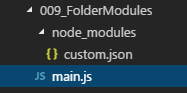
// 1. Если 'имя\_модуля/index' или 'имя\_модуля/index.js' происходит загрузка JavaScript файла

// 2. Если 'имя\_модуля/index.json', происходит парсинг файла и загружается JavaScript объект

// 3. Если 'имя\_модуля/index.node', происходит загрузка бинарного файла

var obj = require('custom'); // Нужно ввести имя подключаемого файла

console.log(obj.name);



Caching

// В node.js при первом подключении модуля происходит выполнение кода в подключаемом модуле, и модуль помещается в кеш

// при повторном подключени того же модуля, среда проверяет путь к подключаемому файлу, если в кеше уже есть такой путь, то возвраoается уже существующий объект

var user = require('./cache')

user.sayHello();

require('./sample');

user.sayHello();

module.exports = {

fname : "Ivan",

lname : "Ivanov",

age : 25,

sayHello : function () {

console.log('Hello! ', 'My name is ', this.fname, ' I\'m ', this.age, ' years old!')

}

}

var test = require('./cache');

test.fname = 'Sergey';

test.age = 30;

# Events

## Emitter

// Для работы с событиями, необходимо подключить модуль 'events'

var evt = require('events');

// Все обьекты которые генерируют события в Node.js должны быть экземплярами класса EventEmitter, или его наследников

var emitter = new evt.EventEmitter();

// Что бы установить слушателя на событие, необходимо воспользоваться методом on(), который доступен на обьекте события

emitter.on('create', function(){

console.log('Folder was created!');

});

// Метод emit предназначен для генерации события

emitter.emit('create');

## Custom events

var evt = require('events');

// Создаем функцию конструктор, которая будет унаследована от функции EventEmitter

// Объект созданый функцией позволит подключиться к файлу и считать из него данные

function ReadFile(){

this.\_file = "";

}

// В прототип ридера записываем объект EventEmitter что бы была возможность генерировать события

ReadFile.prototype = new evt.EventEmitter();

// fileName - имя файла

// callback - функция, которая вызовется после того как данные будут прочитаны

ReadFile.prototype.readDataFromFile = function(path, callback){

this.file = path;

if(typeof callback == 'function'){

this.on('readData', callback)

}

this.\_read();

};

ReadFile.prototype.\_read = function(){

console.log('Loading...');

var someDataFromFile = 'Text text text'; // данные считанные из файла

this.emit('readData', someDataFromFile);

console.log('Data was read.');

}

module.exports.Reader = ReadFile;

// Для работы с событиями, необходимо подключить модуль 'events'

var fileReader = require('./readFile.js');

// Создаем объект ридера

var reader = new fileReader.Reader();

// функция readDataFromFile подключается к файлу, считывает данные и передает в нашу callback функцию

reader.readDataFromFile('file.js', function(responce){

console.log(responce);

});

## Add Listener

'use strict'

var count = 0;

var print = () => {

count++;

console.log('Click - ', count);

}

// Для работы с событиями, необходимо подключить модуль 'events'

var evt = require('events');

var emt = new evt.EventEmitter;

// метод on и addListener добавляют обработчик на событие

emt.on('click', print);

emt.addListener('click', print);

// добавив обработчик с помощью метода once, обраюотчик сработает только один раз и будет удален

emt.once('click', print);

// Вызов обработчиков

emt.emit('click');

emt.emit('click');

## Synchronous

// Обработчики, которые установленны на определенное событие, вызываются синхронно

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

emt.on('event', function(){

console.log("Listener № 1");

});

emt.on('event', function(){

console.log("Listener № 2");

});

emt.on('event', function(){

console.log("Listener № 3");

});

// Генерируем событие myEvent и в функцию обработчик передаем 2 параметра

emt.emit('event');

## Asynchronous

// Обработчики, которые установленны на определенное событие, вызываются синхронно

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

emt.on('event', function(){

setImmediate(() => {

console.log("Listener № 1");

});

});

emt.on('event', function(){

console.log("Listener № 2");

});

emt.on('event', function(){

setImmediate(() => {

console.log("Listener № 3");

});

});

// Генерируем событие myEvent и в функцию обработчик передаем 2 параметра

emt.emit('event');

## Remove Listener

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

// Добавляем обработчик на событие myEvent

emt.on('myEvent', test);

console.log('Listener added!');

// Генерируем событие myEvent

emt.emit('myEvent');

// Удаляем обработчик с события myEvent

emt.removeListener('myEvent', test);

console.log('Listener removed!');

// Повторно генерируем событие myEvent

emt.emit('myEvent');

function test(){

console.log('test function!');

}

## Prepend Listener

// Для работы с событиями, необходимо подключить модуль 'events'

var evt = require('events');

var emt = new evt.EventEmitter;

emt.on('myEvent', function(){

console.log('test string 1');

});

// Метод prependListener добавляет подписчика в начало цепочки

emt.prependListener('myEvent', function(){

console.log('test string 2');

});

emt.once('once', function(){

console.log('First once listener');

});

// Метод prependListener добавляет подписчика в начало цепочки

emt.prependOnceListener('once', function(){

console.log('Seconds once listener');

});

emt.emit('myEvent');

emt.emit('once');

## Loggin event add and remove

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

var count = 0;

// придобавлении нового подписчика, генерируется событие newListener

emt.on('newListener', function(event, listener){

if(event == 'myEvent'){

console.log('myEvent was be added to emt.')

}

});

// когда подписчик удаляется, срабатывает событие removeListener

emt.on('removeListener', function(event, listener){

if(event == 'myEvent'){

console.log('myEvent was be remowed from emt.')

}

});

emt.on('myEvent', test);

console.log(emt.listenerCount('myEvent'));

emt.removeListener('myEvent', test);

console.log(emt.listenerCount('myEvent'));

function test(){

console.log('test event!');

}

## Parametrs

var evt = require('events');

var emitter = new evt.EventEmitter();

emitter.on('click', function(a, b){

console.log('function 1');

console.log(a, b);

});

emitter.on('click', function(){

console.log('function 2');

console.log(arguments);

});

emitter.emit('click', 1, 2);

## Arrow function

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

// Когда вызывается функция обработчик, ключевое слово this внутри нее указывает на объект EventEmitter,

// от этого можно избавиться используя стрелочную функцию

emt.on('myEvent', function(){

console.log('Ordinary function: ');

console.log(this);

});

emt.on('myEvent', () => {

console.log('Arrow function: ')

console.log(this)

});

// Генерируем событие myEvent и в функцию обработчик передаем 2 параметра

emt.emit('myEvent');

## Max Listener

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

// emt.setMaxListeners(20);

for (var index = 0; index < 10; index++) { // попробовать 11

(function(){

var current = index;

emt.on('myEvent', function(){

console.log(current);

});

})();

}

// функция возвразает количество зарегистрированных обработчиков указанного события

console.log('EMT has', emt.listenerCount('myEvent'), 'listeners.');

// Генерируем событие myEvent

emt.emit('myEvent');

## Listeners

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

emt.on('myEvent', function(){

console.log('First listener.');

});

emt.on('myEvent', function(){

console.log('Second listener.');

});

// метод listeners возвращает массив функций обработчиков для указаного события

var listeners = emt.listeners('myEvent');

for (var index = 0; index < listeners.length; index++) {

listeners[index]();

}

# Console

## Enviroment

### Cache

require('./test.js');

require('./simple.js');

console.log('Main file is loaded!');

console.log('');

console.log(require.cache); // свойство cache объекта require, хранит закешированные модули

### FileInfo

// \_\_dirname - глобальная переменная, хранит имя текущей директории

console.log("Directory name:", \_\_dirname);

// \_\_filename - глобальная переменная, хранит имя текущего файла

console.log("File name:", \_\_filename);

### Path1

// path модуль ядра, обеспечивающий обработку и преобразование путей к файлам

var path = require('path');

// С помощью метода basename можно получить имя файла с расширением

var filename = path.basename(\_\_filename);

console.log('Current file: ', filename);

// path.resolve - метод преобразующий относительный путь в абсолютный

console.log(path.resolve('./test'));

// метод возвразает расширение файла

var extension = path.extname(\_\_filename);

console.log('Current file, has', extension, 'extensions!');

// path.isAbsolute проверяет, является ли путь абсолютным

var abs = path.isAbsolute(\_\_dirname);

console.log('path.isAbsolute(', \_\_dirname, ') - ', abs);

console.log('path.isAbsolute(','public/myProject/test', ') - ', path.isAbsolute('public/myProject/test'));

### Path2

// path модуль ядра, обеспечивающий работу с путями директорий и файлов

var path = require('path');

var file = '004\_Path\_2.js';

// метод join предназначен для генерации путей, на основе принятых параметров

var filePath = path.join(\_\_dirname, file);

console.log(filePath);

// Парсит заданый путь и возвращает объект

var pathParts = path.parse(filePath);

console.log(pathParts);

console.log('Systems separator:', path.sep);

var objFormated = {

root: 'D:\\',

dir: 'D:\\Node\\003\_ConosleProgram\\001\_Environment',

base: '004\_Path\_2.js',

ext: '.js',

name: '004\_Path\_2'

}

// Метод format генерирует новый путь на основе свойств объекта

var obj = path.format(objFormated);

console.log(obj);

### Process

// Объект process является глобальным и всегда доступен, без подключения модулей.

// Этот объект предоставляет информацию и позволяет управлять текущим процессом

// Свойство хранит информацию о архитектуре процессора

console.log(process.arch);

// Свойство хранит путь по которому запускается Node js, путь по которому находится исполняемый файл, и аргументы командной строки

console.log(process.argv);

// возвразает путь к серверу

console.log(process.execPath);

// метод возвращает версию Node js

console.log(process.version);

// метод возвращает название платформы, на которой выполняется приложение

console.log(process.platform);

// возвращает информацию об использовании памяти

console.log(process.memoryUsage());

// {

// heapTotal: 1826816, - используемая движком V8, память

// heapUsed: 650472, - используемая движком V8, память

// external: 49879 - память используемая внешними библиотеками

// }

## Utils

### Inherits

var utils = require('util');

function Base(){

this.name = "Base function"

}

Base.prototype.say = function(){

console.log('Hello, this is a %s function', this.name);

}

function Derived(){

this.name = "Derived";

}

// Метод utils.inherits позволяет производить наследование

utils.inherits(Derived, Base);

Derived.prototype.getData = function(){

console.log('Some data from Derived function')

}

var derived = new Derived();

derived.getData();

derived.say();

### Format

var utils = require('util');

// %s - Строка

// %d - Число (целое или число с плавающей запятой)

// %j - JSON

// %% - символ '%'

var user = {

name : "Ivan",

age : 25,

salary : 10000,

bonus: 15

}

var str = utils.format('Hello, my name is %s. I\'m %d years old! My bonuses from the salary are %d%%', user.name, user.age, user.bonus);

console.log(str);

console.log();

console.log('%j', user);

### Inspect

var utils = require('util');

var user = {

name : "Ivan",

age : 25,

salary : 10000,

bonus: 15

}

var objInfo = utils.inspect(user);

console.log(objInfo);

var point = {

name : "A",

x : 10,

y : 20,

inspect : function(){

return utils.format('Point %s[%d:%d]', this.name, this.x, this.y);

}

}

console.log(point);

### Check type

var utils = require('util');

var obj = { name: "Some object" };

var numb = 10;

var str = "Some string";

var und = null;

var arr = [1,2,3,4,5];

console.log('arr is array:', utils.isArray(arr));

console.log('obj is object:', utils.isObject(obj));

console.log('numb is number:', utils.isNumber(numb));

console.log('str is string:', utils.isString(str));

console.log('und is NULL:', utils.isNull(und));

console.log('und is function:', utils.isFunction(und));

## Console

### Console

// Объект Console, является глобальным и обеспечивает простую отладку приложений в Node.js

// По умолчинию, объект console сконфигурирован на передачу информации в\из process.stdout и process.stdin

console.log('Simple message.');

console.info('Some data'); // алиас функции console.log()

console.error(new Error('Somthig wrong!'));

var name = "---";

console.warn('Danger ${name}!'); // алиас функции console.error()

var obj = {name : "Ivan", age : 25};

console.dir(obj);

### Console time

var arr = [];

// функция console.time(label) позволяет запустить таймер, для замера продолжительности выполнения операции.

console.time('1-st');

for (var index = 0; index < 1000000; index++) {

arr[index] = (index + 1) \* 2;

}

// функция console.timeEnd(label) останавливает таймер который был вызван функцией console.time(label)

console.timeEnd('1-st'); // Время в миллисекундах

### Buffer

// Класс Buffer позволяет работать с потоками двоичных данных.

// Он является глобальным объектом, поэтому не нужно использовать функцию require

// Метод Buffer.alloc - создает буфер, проинициализированный нулями на указанное количество байт

var buff1 = Buffer.alloc(10);

console.log('empty buffer:', buff1);

// Метод Buffer.allocUnsafe создает непроинициализированный буфер

var nibuff = Buffer.allocUnsafe(10);

console.log('uninitialized buffer:', nibuff);

console.log('Lenght of buffer:', buff1.length);

console.log(Buffer.byteLength('hello world', 'utf-8'));

var buff1 = Buffer.from([10, 20, 30, 40, 50, 60, 70, 80, 90]);

var buff2 = Buffer.from('Hello world!!!', 'utf8');

// Возможные значения кодировок

// 'ascii' - только для 7-битных ASCII-cтpoк. Этот метод кодирования очень быстрый, он сбрасывает старший бит символа;

// 'utf' - Uniсоdе-символы UTF-8;

// 'binary' - хранит двоичные данные в строке, используя младшие 8 бит каждого символа.

// 'base64' - строка, закодированная в системе Base64;

// 'hex' - кодирует каждый байт как два шестнадцатеричных символа

var buff3 = Buffer.from([50, 60, 70, 80, 90, 10, 20, 30, 40]);

console.log(buff1[2]);

// buf.compare(target[, targetStart[, targetEnd[, sourceStart[, sourceEnd]]]])

// метод сравнивает два буфера, и возвращает 0 - если буферы равны, 1 если buf > target, -1 если buf < target

console.log('Compare buffer:', buff1.compare(buff2));

console.log('index compare:', buff1.compare(buff3, 5, 9, 0, 4));

// Buffer.concat метод объединяющий несколько буферов в один

var newBuff = Buffer.concat([buff1, buff2]);

console.log(newBuff);

// метод проверяет является ли параметр типом Buffer

console.log(Buffer.isBuffer(newBuff));

var symb = buff2.indexOf('l');

console.log('Position of symbol \'l\' from start', symb);

symb = buff2.lastIndexOf('l');

console.log('Position of symbol \'l\' from end', symb);

// Типизированный массив, который хранит 16 разрядные значения

var arr1 = new Uint16Array(11);

// свойство buffer возвращает буфер на который указывает массив

const buf1 = Buffer.from(arr1.buffer);

var arr2 = new Uint16Array(4);

const buf2 = Buffer.from(arr2.buffer);

var arr4 = new Uint16Array(10);

const buf4 = Buffer.from(arr4.buffer);

// Метод buf.write(string[, offset][, length][, encoding]) - запись данных в буфер, аргументы:

// string - данные в строковом виде

// offset - индекс буфера, с которого начнется запись

// length - количество байтов для записи

// encoding - кодировка; по умолчанию utf-8

buf1.write('123', 2, 2);

buf2.write('1234', 2, 3);

// buf.toString([encoding][, start][, end]) - чтение данных из буфера, аргументы:

// encoding - кодировка; по умолчанию utf-8

// start - индекс, с которого начать чтение данных буфера

// end - индекс, до которого читать данные буфера

var bufData = buf1.toString('utf-8', 2, 3);

console.log(bufData); // 1

// конкатенация буферов: Buffer.concat(list[, totalLength]), аргументы:

// list - список буферов для конкатенации

// totalLength - длина буфера, получаемого в результате конкатенации

var buf3 = Buffer.concat([buf1, buf2], 28);

console.log(buf3.toString()); //12

// копирование буферов: buf.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd]), аргументы:

// targetBuffer - буфер, в который копировать данные

buf2.copy(buf4, 0, 0, 3);

console.log(buf4.toString());

## File Stream

### Fs

var fs = require('fs');

console.log("Going to open file!");

// открыть файл

// open(filename, flag, callback)

// filename - имя файла

// flag - флаг, который указывает каким образом открывать файлы

// r - открыть для чтения. Генерирует исключение при отсутствии файла;

//r+ - открыть для чтения и записи. Генерирует исключение при отсутствии файла;

//rs - открыть для чтения в синхронном режиме;

//rs+ - открыть для чтения и записи в синхронном режиме;

//w - открыть для записи. Если файл не существует, он будет создан. Если файл существует, его содержимое будет очищено;

//w+ - открыть для чтения и записи. Если файл не существует, он будет создан. Если файл существует, его содержимое будет очищено;

//а - открыть для записи в конец файла. Если файл не существует, он будет создан;

//а+ - открыть для чтения и записи в конец файла. Если файл не существует, он будет создан

// callback - функция, которая вызывается после завершения чтения

fs.open('demofile.txt', 'w+', function (err, fd) {

console.log('opening file!');

if (err) {

console.log(err);

}

else {

// Метод write - альтернатива методу writeFile; позволяет записать данные в файл и принимает такие аргументы:

// fd - дескриптор файла

// buffer - данные в виде буфера или строки,

// offset, length - определяют часть буфера, которую следует записать в файл

// position - отступ от начала файла, куда будут записаны данные

// callback - функция, принимающая аргументы err, written, string

fs.write(fd, 'This is the file content!', { encoding: 'utf-8' }, function (err, written, string) {

console.log('writing to file!');

if (err) throw err;

console.log(written); // written - количество байтов, которое потребовалось для записи данных

console.log(string); // строка, записанная в файл

});

var arr = new Uint16Array(1024);

var buf = Buffer.from(arr.buffer);

// Метод read позволяет читать данные из файла, принимает аргументы:

// fd -дескриптор файла

// buffer - буфер, в который будут помещены прочитанные данные

// offset, length - определяют часть буфера, которую следует записать в файл

// position - отступ от начала файла, данные которого считываются

// callback - функция, принимающая аргументы err, bytesRead, buffer

fs.read(fd, buf, 0, buf.length, 0, function (err, bytes) {

console.log('reading from file!');

if (err) throw err;

console.log(bytes);

console.log(buf.slice(0, bytes).toString());

})

// закрыть файл

fs.close(fd, function (err) {

if (err) throw err;

console.log('file closed!');

})

}

});

### writeFile

// Модуль fs предназначен для работы с файлами. методы содержащиеся в модуле имеют синхронную и асинхронную формы

var fs = require('fs');

var utils = require('util');

user = {

fname : "Ivan",

lname : "Ivanov",

age : 30,

position : "developer"

}

console.log('File writing...');

fs.writeFile('text.txt', utils.format('%j', user), function(err){

if(err){

console.log(err);

return;

}

console.log('File was wrote!');

});

### readFile

var fs = require('fs');

fs.exists('text.txt', function(){

fs.readFile('text.txt', {encoding : 'utf-8'}, function(err, data){

if(err){

console.log(err);

return;

}

console.log("it's- "+ data);

var obj = JSON.parse(data);

console.log(obj.fname, obj.lname, obj.age, obj.position);

});

});

### readDir

var fs = require('fs');

fs.readdir('testdir', function(err, filenames){

console.log(filenames);

});

### Watcher

var fs = require('fs');

fs.watch('text.txt', function(event, filename){

console.log('File %s is %s.', filename, event);

});

fs.writeFile('text.txt', 'test string', function(err){

if(err) throw err;

console.log('data was wrote');

});

## Stream

### Read and write stream

// Потоки(Streams) - объекты, которые позволяют считывать данные или записывать данные в источник в непрерывном режиме.

// В NodeJS есть 4 типа потоков:

// Readable - поток, который используется для операций чтения

// Writable - поток, который используется для операци записи

// Duplex - поток, который может быть использован и для операций чтения, и для операций записи

// Transform - тип duplex-потока, в котором выходные данные рассчитываеются на основе входных данных

var fs = require('fs');

var writeData = 'This is the file content!';

// поток для записи данных

// генерирует события error(при ошибке), finish(при завершении текущей операции записи данных)

var writerStream = fs.createWriteStream('output.txt');

writerStream.write(writeData, 'utf-8');

writerStream.end();

writerStream.on('finish', function() {

console.log('Write completed');

});

writerStream.on('error', function(err) {

console.log(err);

});

// поток для чтения данных

// генерирует события error(при ошибке), data(когда данные доступны для чтения),

// end(когда больше нет доступных данных для чтения)

var readData = '';

var readerStream = fs.createReadStream('output.txt');

readerStream.setEncoding('UTF8');

readerStream.on('data', function (chunk) {

readData += chunk;

});

readerStream.on('end', function () {

console.log(readData);

});

readerStream.on('error', function (err) {

console.log(err);

});

### Pipe stream

// Piping streams - перенаправление данных одного потока к другому потоку

var fs = require("fs");

// создать поток для чтения данных

var readerStream = fs.createReadStream('input.txt');

// создать поток для записи данных

var writerStream = fs.createWriteStream('output.txt');

// передача данных потока readerStream потоку writerStream

readerStream.pipe(writerStream);

writerStream.on('finish', function () {

console.log('data written to file output.txt')

});

## Mock url

var url = require('url');

// Строка URL - структурированная строка, состоящая из многочисленных значимых компонентов.

// При применении к ней метода parse возвращается объект URL, содержащий все эти компоненты в качестве свойств

var mockUrl = 'http://user:pass@host.com:8080/p/a/t/h?query=string#hash';

// при вызове url.parse со вторым параметром true

// функция будет парсить параметры запроса(queryString)

var mockUrlObj = url.parse(mockUrl);

console.log(mockUrlObj);

// метод url.format преобразует объект URL в строку URL

var formattedUrl = url.format(mockUrlObj);

console.log(formattedUrl);

// метод url.resolve(from, to) преобразует второй аргумент(целевой URL) по образцу первого аргумента(базовый URL)

var resolvedUrl = url.resolve('http://example.com/one/two', '/two/three');

console.log(resolvedUrl); // 'http://example.com/two/three'

## Timers

// setTimeout(callback, delay[, ...arg])

// callback - функция, которая будет вызвана через указанный интервал времени

// delay - задержка (в миллисекундах)

// ...arg - аргументы, которые будут использованы при вызове callback

var timeout = setTimeout(function () {

console.log('timeout example');

}, 2000)

// clearTimeout(timeout); // отмена setTimeout

// setInterval(callback, delay[, ...arg])

// callback - функция, которая будет вызвана через указанный интервал времени

// delay - задержка (в миллисекундах)

// ...arg - аргументы, которые будут использованы при вызове callback

var counter = 0;

var interval = setInterval(function () {

console.log('interval ' + counter);

counter++;

if (counter > 10) {

clearInterval(interval); // отмена setInterval

}

}, 100)

# Http

## createServer

var http = require('http');

var server = http.createServer();

var server = http.Server();

server.on('request', function(request, response){

console.log('Connection established!');

console.log(request.method);

// Завершает конфигурацию ответ

response.end();

// response.end("Responce is configured!", 'utf8', () => { console.log('finish'); });

});

server.listen(8080)

server.on('listening', function(){

console.log('Server running on port 8080');

});

## write and writeHead

var http = require("http");

var server = http.createServer(function(request, response) {

// writeHead - метод позволяет записать в ответ заголовки и статус код

response.writeHead(200, {"Content-Type": "text/plain"});

// write - метод позволяет создать тело ответа в виде потока writeable Stream

response.write("Hello World");

// end - завершает конфигурациюответа и отправляет его

response.end();

}).listen(8080, function(){

console.log('Server running on port 8080');

});

## request

var http = require('http');

var utils = require('util');

http.createServer(function(req, res){

var requestInfo = utils.format('HTTPVersion: %s \nMethod: %s \nStatus code: %s \nMessage: %s \nURL: %s',

// верисия http протокола

req.httpVersion,

// http глагол

req.method,

// статус код

req.statusCode,

// текстовое описание статус кода

req.statusMessage,

// запрашиваемый ресурс

req.url);

console.log(requestInfo);

console.log();

// headers - свойство содержит объект с заголовками

for (var key in req.headers) {

console.log(key, ":", req.headers[key]);

}

// отправляем ответ клиенту

res.end();

}).listen(8080, 'localhost');

## headers

var http = require('http');

http.createServer(function(req, res){

// возвращает массив c заголовками, четные элементы массива - имена заголовков, нечетные - значения заголовков

console.log(req.rawHeaders);

// setHeader - метод записывает в ответ указаный заголовок и его значение

res.setHeader('Content-Type', 'text/plain');

//

var ct = res.getHeader('Content-Type');

console.log(ct);

// свойство определяет будет ли сервр отправлять заголовок Date. Если значение свойства false - дата не отправляется

res.sendDate = false;

//

console.log(res.headersSent);

res.end('<h1>Test page</h1>');

console.log(res.headersSent);

}).listen(8080);

## returnPage

var http = require('http');

var url = require('url');

var fs = require('fs');

http.createServer(function(req, res){

var path = url.parse(req.url).pathname;

path = '005\_returnPage/' + path.substr(1);

console.log(path);

fs.readFile(path, function(err, data){

if(err){

console.log(err);

res.writeHead(404, { 'Content-Type' : 'text/plain'});

res.end('Not Found!');

}

else{

res.writeHead(200, { 'Content-Type' : 'text/html'});

res.write(data.toString());

console.log(data.toString());

console.log('data was sent');

res.end();

}

});

}).listen(8080, function(){

console.log('Server starting!');

});

## queryParam

const http = require('http');

const url = require('url');

const server = http.createServer();

var port = 8080;

server.on('request', function(req, res) {

var method = req.method;

var \_url = req.url;

    console.log('Method:', method, '; URL:', \_url);

    var parsed = url.parse(req.url, true);

    console.log(parsed);

    if (parsed.pathname == '/test' && parsed.query.message) {

        res.statusCode = 200; // OK

        res.end(parsed.query.message);

    }

    else {

        res.statusCode = 404; // Not Found

        res.end('Page not found on server!');

    }

});

server.listen(port);

server.on('listening', function() {

    console.log('Server running on port ' + port);

})

## get

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" />

<title>Just a Page</title>

<script>

window.onload = function() {

    var btn = document.getElementById('btn');

    var container = document.getElementById('output');

        // функция для отправки GET запроса при нажатии на кнопку

    function makeRequest() {

        var xhr = new XMLHttpRequest();

        xhr.open('GET', 'request');

        xhr.onload = function() {

                container.innerHTML += this.responseText;

        };

        xhr.onerror = function() {

                console.log('error!');

        };

        xhr.send();

    };

    btn.addEventListener('click', makeRequest);

};

</script>

</head>

<body>

<button class="btn-lg btn btn-success" id="btn">Get Data!!!</button>

<div id="output"></div>

</body>

</html>

var http = require('http');

var fs = require('fs');

var url = require('url');

var path = require('path');

var port = 8080;

const server = http.createServer(function(req, res) {

    // обработка ошибок запросов

    req.on('error', function(err) {

       console.log(err);

    });

console.log(req);

if (req.url == '/') {

// чтение файла index.html

var file\_path = path.join(\_\_dirname, '009\_index.html');

fs.readFile(file\_path, function (err, data) {

// обработка ошибок

if (err) {

console.log(err);

res.writeHead(404, { 'Content-Type': 'text/plain' });

res.write('Not Found!');

} else {

res.writeHead(200, { 'Content-Type': 'text/html' });

// записать в овет содержимое читаемого файла

res.write(data.toString());

}

res.end();

});

}

else if (req.url == '/request') {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.write('<h2>Data from server!</h2>');

res.end();

console.log('data sent!');

}

else {

res.writeHead(404, { 'Content-Type': 'text/html' });

res.end('Resource not found');

}

}).listen(port);

console.log('server running on port ' + port);

## post

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" />

<title>Just a Page</title>

<script>

window.onload = function() {

    var btn = document.getElementById('btn');

    var container = document.getElementById('output');

    function makeRequest(data) {

        var xhr = new XMLHttpRequest();

        xhr.open('POST', 'data');

        xhr.onload = function() {

var data = JSON.parse(this.responseText);

            for (var i = 0; i < data.length; i++) {

container.innerHTML += `Product: ${data[i].name} - ${data[i].price}$ <br/>`;

}

        };

        xhr.onerror = function() {

            console.log('error!!!');

        };

        xhr.send(data);

    };

    btn.addEventListener('click', function() {

var data = '<h3>This is some data!</h3>';

makeRequest(data);

});

};

</script>

</head>

<body>

<button class="btn-lg btn btn-success" id="btn">Get Data!!!</button>

<div id="output"></div>

</body>

</html>

var http = require('http');

var fs = require('fs');

var url = require('url');

var path = require('path');

var port = 8080;

var data = [

{name : 'Book', price : 10},

{name : 'Pen', price : 20},

{name : 'Lamp', price : 30},

{name : 'Pencil', price : 40},

{name : 'Desk', price : 50}

];

var server = http.createServer(function(req, res) {

// обработка ошибок запросв

req.on('error', function (err) {

console.log(err);

});

if (req.url == "/") {

// чтение файла index.html

var file\_path = path.join(\_\_dirname, '010\_index.html');

fs.readFile(file\_path, function (err, data) {

// обработка ошибок

if (err) {

console.log(err);

res.writeHead(404, { 'Content-Type': 'text/plain' });

res.write('Not Found!');

} else {

res.writeHead(200, { 'Content-Type': 'text/html' });

// записать в овет содержимое читаемого файла

res.write(data.toString());

}

res.end();

})

} else if (req.url == "/data") {

var body = '';

// получение данных POST запроса

req.on('data', function () {

body = data.toString();

// создание тела ответа

res.writeHead(200, { 'Content-Type': 'application/json' });

res.write(JSON.stringify(data));

res.end();

console.log('data sent!');

});

} else {

res.writeHead(404, { 'Content-Type': 'text/html' });

res.end('Resource not found');

}

}).listen(port);

console.log('server running on port ' + port);

## getHandle

// сервер для обработки запроса

var server = http.createServer(function (req, res) {

console.log('request')

res.end('GET request path: ' + req.url);

}).listen(port);

## postHandle

var http = require('http');

var fs = require('fs');

var url = require('url');

var port = 8080;

var server = http.createServer(function(req, res) {

console.log(res);

     var body = '';

    // обработка ошибок запросв

    req.on('error', function(err) {

       console.log(err);

    });

    // получение данных POST запроса

    req.on('data', function(data) {

       body = data.toString();

       // создание тела ответа

       res.writeHead(200, {'Content-Type': 'text/html'});

        res.write(body);

        res.end();

        console.log(`data from request: ${body}`);

    });

}).listen(port);

## request method

var http = require('http');

http.createServer(function (req, res) {

// формирование разных ответов сервера для запросов с различными HTTP методами

switch (req.method) {

case 'GET': {

var response\_text = 'GET request to path ' + req.url

console.log(response\_text);

res.end(response\_text);

break;

}

case 'POST': {

var response\_text = 'POST request to path ' + req.url

console.log(response\_text);

res.end(response\_text)

break;

}

}

}).listen(8080);

# Express

## simple

// создаем приложение, которое будет принимать запросы, обрабатывать их, и отправлять ответы

var app = express();

// обработчик, который будет срабатывать на get запросы, для маршрута '/'

app.get('/', function(request, response){

console.log(request.url);

response.send('<h1>Hello, world!</h1>');

});

app.get('/about', function(request, response){

console.log(request.url);

response.send('<h1>About Page</h1>');

});

app.get('/products', function(request, response){

console.log(request.url);

response.send('<h1>Products Page</h1>');

});

app.listen(8080);

## route pattern

var express = require('express');

var app = express();

// в качестве маршрута можно задать регулярное выражение, которому должен соответствовать маршрут

app.get('/[a-zA-Z]\*\.html$/', function(request, response){

response.send(request.url);

response.end();

});

app.listen(8080, function(){

console.log('Server started on port: 8080');

})

## acceptable

var express = require('express');

var app = express();

var port = 8080;

// метод all позволяет установить обработчик на маршруты, при обращении к которым протокол не имеет значения

app.all('\*', function(req, res) {

// http метод => GET

    console.log('method: ' + req.method);

// параметры адресной строки в виде объекта

    console.log('query: ' + req.query);

// протокол (http или https)

    console.log('protocol: ' + req.protocol);

// true или false(true если req.protocol == 'https')

    console.log('secure: ' + req.secure);

    // req.accepts - метод, который проверяет типы данных (MIME type), которые допустимы

    // для данного запроса. Если тип данных недопустим, возвращает false

    console.log('accepts: ' + req.accepts(['text/html', 'application/json']));

// возвращает указанный заголовок запроса

console.log('content type header: ' + req.get('Content-Type'));

console.log('------------------------');

    res.end();

});

app.listen(port, function() {

    console.log('app running on port ' + port);

});

## response cookie

var express = require('express');

var http = require('http');

var path = require('path');

var app = express();

var port = 8080;

app.get('/', function(req, res) {

    // метод cookie позволяет создавать cookies

    res.cookie('someCookie', 'this is a cookie', {

        httpOnly: true,

maxAge: 2000

    });

    res.cookie('anotherCookie', 'this is another cookie!');

    // удалить cookie

    res.clearCookie('anotherCookie');

    // метод sendFile позволяет указывать в ответе файл для чтения браузером

    res.sendFile(path.join(\_\_dirname,'/data/007\_index.html'));

});

app.listen(port, function() {

    console.log('app running on port ' + port);

});

## set headers

var express = require('express');

var app = express();

var port = 8080;

app.get('/', function(req, res) {

// res.locals = свойство, хранящее локальные переменные данного цикла request/response

    res.locals.prop1 = 'this is a response property!';

    console.log(res.locals.prop1);

    // метод res.append позволяет установить заголовки ответа

    res.append('Set-Cookie', 'foo=bar; Path=/; HttpOnly');

    res.append('Warning', '199 Miscellaneous warning');

    // метод res.set делает то же самое

    res.set('Cache-Control', 'no-cache');

res.send('<h1>Sample Response</h1>' + '<h3>' + res.locals.prop1 + '</h3>');

});

app.listen(port, function() {

    console.log('app running on port ' + port);

});

## route params

var express = require('express');

var app = express();

// В маршрутах допустимо задавать параметры, для этого необходимо в маршруте определить сегмент, в котором будет находится параметр

// для передачи параметра в сегменте, необходимо перед именем параметра установить двоеточие (:)

// http://example.com/home/index.html?category=notebooks&product=100

// http://example.com/home/index/notebooks/100

app.get('/category/:categoryId', function(request, response){

// для получения параметра из сегмента используется сойство params объекта request

console.log(request.params);

});

app.get('/category/:categoryId/product/:productId', function(request, response){

console.log(`category: ${request.params['categoryId']}`);

console.log(`product: ${request.params['productId']}`);

});

app.listen(8080);

## middleware next

// при получении запроса от клиента, запрос передается в конвеер обработки запросов.

// Конвеер обработки состоит из компонентов, которые в терминологии express называются Middleware

var express = require('express');

var app = express();

// для регистрации middleware используется функция use, all, app.METHOD()

// 1. request - данные запроса

// 2. response - объект для управления ответом

// 3. next - следующая в конвейере обработки функция

app.use('/', function(request, response, next){

console.log('Prehandler...');

// передаем управление следующему обработчику

next();

});

app.get('/', function(request, response){

console.log('Main handler');

// завершаем ответ от сервера

response.end();

});

app.get('/about', function(request, response){

console.log('About handler');

// завершаем ответ от сервера

response.end();

});

app.listen(8080, function(){

console.log('Server start')

});

## cookie parser

// при получении запроса от клиента, запрос передается в конвеер обработки запросов.

// Конвеер обработки состоит из компонентов, которые в терминологии express называются Middleware

var express = require('express');

var app = express();

// для регистрации middleware используется функция use

// 1. request - данные запроса

// 2. response - объект для управления ответом

// 3. next - следующая в конвейере обработки функция

var cookieParser = {

getCookie: function (req, res, next) {

var cookies = req.get('Cookie');

console.log("all -", cookies);

var cookieCollection = cookies.split(';');

var tempCookies = {};

for (var i = 0; i < cookieCollection.length; i++) {

var elem = cookieCollection[i];

// foo = bar

var pos = elem.indexOf('=');

var key, value;

if (pos != -1) {

key = elem.substring(0, pos);

value = elem.substring(pos + 1);

}

tempCookies[key] = decodeURIComponent(value);

}

req.cookies = tempCookies;

next();

}

}

app.use('/', cookieParser.getCookie);

app.get('/', function(req, res){

console.log(req.cookies)

})

app.get('/index', function (req, res) {

console.log('Main handler');

res.cookie('someCookie', 'this is another cookie!');

res.cookie('anotherCookie', 'this is another cookie!');

res.end();

});

app.listen(8080, function () {

console.log('Server start')

});

## logger

var express = require('express');

var fs = require('fs');

var app = express();

var path = 'logger.txt';

app.use(function(request, response){

var data = `Address : ${request.ip}; Time: ${new Date().toLocaleString()}; URL : ${request.url}\n`;

fs.appendFile(path, data, function(err){

console.log('data wrote');

});

});

app.get('/', function(request, response){

console.log('Main handler');

response.end();

});

app.listen(8080, function(){

console.log('Server start')

});

## class router

var express = require('express');

var app = express();

// Класс Router позволяет определить маршрут, в пределах которого можно создавать подмаршруты и задавать им обработчики

var router = express.Router();

router.route("/")

.get(function(req, res){

res.send("List of products. Get method.");

})

.post(function(req, res){

res.send("Product created. POST method.");

});

router.route("/:id")

.get(function(req, res){

res.send(`Product ${req.params.id}`);

});

app.use("/products", router);

app.get("/", function(req, res){

res.send("Главная страница");

});

app.listen(8080);

## static files

var express = require('express');

var path = require('path');

var app = express();

var catalog = 'data/012\_example';

// компонент express.static() указывает на каталог с файлами

app.use('/', express.static(path.join(\_\_dirname ,catalog)));

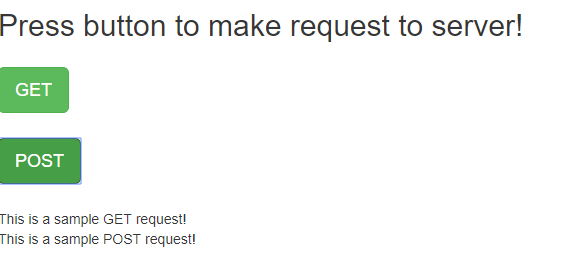
// app.get('/', function(req, res){

// res.sendFile(path.join(\_\_dirname, catalog, 'index.html'));

// });

app.listen(8080);

## get and post request



<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" />

<title>Make Requests!</title>

<style>

div {

    margin-top: 25px;

}

</style>

<script>

window.onload = function() {

    var btns = document.getElementsByTagName('button');

    // отправка GET запроса

    btns[0].addEventListener('click', function() {

        var getReq = new XMLHttpRequest();

        var str = encodeURIComponent('This is a sample GET request! <br />');

        getReq.open('GET',`/test?text=${str}`);

        getReq.onload = function() {

            document.getElementById('output').innerHTML += this.responseText;

        };

        getReq.send();

    });

    // отправка POST запроса

    btns[1].addEventListener('click', function() {

        var postReq = new XMLHttpRequest();

        postReq.open('POST','/test');

        postReq.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

        postReq.onload = function() {

            document.getElementById('output').innerHTML += this.responseText;

        };

        var str = encodeURIComponent('This is a sample POST request! <br />');

        postReq.send(`text=${str}`);

    });

}

</script>

</head>

<body>

<h2>Press button to make request to server!</h2>

    <div><button class="btn-success btn btn-lg">GET</button></div>

    <div><button class="btn-success btn btn-lg">POST</button></div>

    <div id="output"></div>

</body>

</html>

var express = require('express');

// модуль оторы позволяет распарсить данные входящего запроса и поместить их в req.body

var bodyParser = require('body-parser');

var url = require('url');

var app = express();

var port = 8080;

// static - middleware функция для работы с файлами

app.use(express.static('data/013\_example'));

// middleware для обработки тела запроса в кодировке urlencoded

app.use(bodyParser.urlencoded({ extended: true }));

//Метод app.route() позволяет создавать обработчики маршрутов

app.route('/')

    .all(function(req, res) {

        console.log('request to main page!');

        // перенаправление на страниу index.html

        res.writeHead(301, {'Location':'index.html'});

        res.end();

});

app.route('/test')

    .get(function(req, res) {

        // чтение параметров GET запроса

        var data = url.parse(req.url, true).query;

        res.writeHead(200, {'Content-Type':'text/html'});

        res.end(data.text);

    })

    .post(function(req, res) {

        // чтение данных POST запроса

        var data = req.body.text;

        res.writeHead(200, {'Content-Type':'text/html'});

        res.end(data);

    })

app.listen(port, function() {

    console.log('app running on port ' + port);

});

## sub app

var express = require('express');

var port = 8080;

// вложенные приложения используются для маршрутизации

var app = express(); // главное приложение

var admin = express(); // вложенное приложение

var user = express();

app.get('/', function(req, res) {

    res.writeHead(200, {'Content-Type':'text/html'});

    res.write('<a href="/admin">admin page</a>');

    res.write('<br />');

    res.write('<a href="/user">user page</a>');

    res.end();

});

admin.get('/', function (req, res) {

// свойство mountpath содержить текущий путь маршрутизации(route)

console.log(admin.mountpath); // /admin

res.send('Admin Homepage');

});

// событие mount генерируется, когда происходит привязка дочернего(вложенного)

// приложения к родительскому

admin.on('mount', function() {

    console.log('admin mounted');

});

user.get('/', function (req, res) {

    console.log(user.mountpath); // /user

    res.send('User Homepage');

});

user.on('mount', function() {

    console.log('user mounted');

});

// связывание главного приложения со вложенным

app.use('/admin', admin);

app.use('/user', user);

app.listen(port, function() {

    console.log('app running on port ' + port);

});

## error handle

var express = require('express');

var app = express();

var port = 8080;

app.use(function(req, res, next) {

    if (req.url == '/') {

        res.send('Hello');

    } else {

        next();

    }

});

app.use(function(req, res, next) {

    if (req.url == '/forbidden') {

        next(new Error('access denied!'));

    } else {

        next();

    }

});

app.use(function(req, res, next) {

    if (req.url == '/test') {

        res.send('Test');

    } else {

        next();

    }

});

app.use(function(req, res, next) {

    next(new Error('page not found!'));

})

// если у middleware функции 4 аргумента, express воспринимает ее как обработчик ошибок

app.use(function(err, req, res, next) {

    res.status(500).send('Ooops...Something went wrong!');

    next(err.message);

});

app.listen(port, function() {

    console.log('app listening on port ' + port);

});

# Databases

## Mongodb

CMD -> mongod // Starting server

CMD -> mongo //shell for work with mongo

### connect

// для работы с mongodb необходимо подключить модуль. Для этого используйте комманду - npm i mongodb

// MongoClient основной клас для работы с БД, через него происходят все взаимодействия с БД

var MongoClient = require('mongodb').MongoClient;

var format = require('util').format;

// Путь, по которому устанавливается соединение c БД, test - имя базы к которой подключаемся

var url = 'mongodb://localhost:27017/test';

// Используем метод connect для подключения к серверу, функция которая передается в метод, принимает два параетра

// err - ошибка, которая возникла при установке соединения с БД

// db - ссылка на объект БД

MongoClient.connect(url, function(err, db) {

if(err) throw err;

console.log('Connection established!');

// Закрываем подключение с БД

db.close();

});

### insert

var MongoClient = require('mongodb').MongoClient;

var url = "mongodb://localhost:27017/userCollectionDB";

MongoClient.connect(url, function(err, db){

// в mongoDB нет таблиц, все данные хранятся в коллекциях, и для того что бы работать с БД необходимо получить объект коллекции

// метод collection применяется для получения объекта коллекции

var collection = db.collection('users');

var user = {firstName : "Ivan", lastName : "Ivanov", age: 30};

// метод insertOne позволяет добавить новый объект (документ) в коллекцию, принимает два параметра:

// 1-й - объект оторый необходимо добавить

// 2-й - функция обратного вызова которая выполнится после добавления объекта

collection.insertOne(user, function(err, result){

if(err){

console.log(err);

return;

}

// result.ops - объект полученный обратно из базы данных, содержащий идентификатор

console.log(result.ops);

db.close();

});

});

### insert many

var MongoClient = require('mongodb').MongoClient;

var persons = require('./persons');

var url = 'mongodb://localhost:27017/userCollectionDB1';

MongoClient.connect(url, function(err, db){

var collection = db.collection('users');

// метод insertMany используется для добавления множества обьектов

collection.insertMany(persons, function(err, results){

if(err) throw err;

console.log('Data added!');

console.log('\*\*\*\*\*\*\*\*\*\* Result \*\*\*\*\*\*\*\*\*\*');

console.log(results);

console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

db.close();

});

});

### find

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost:27017/userCollectionDB1';

MongoClient.connect(url, function(err, db){

var collection = db.collection('users');

// Метод find возвращает специальный объект - Cursor

var cursor = collection.find();

// Метод toArray - все полученные данные преобразовывает в массив и передает в функцию обратного вызова

cursor.toArray(function(err, res){

console.log('\*\*\*\*\*\*\*\*\*\* All Data \*\*\*\*\*\*\*\*\*\*');

console.log(res);

console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

// Метод find(параметр) - возвращает коллекцию объектов, которые соответсвуют условию

collection.find({name : "Sergey"}).toArray(function(err, res){

console.log('\*\*\*\*\*\*\*\*\*\* Employees with name Sergey \*\*\*\*\*\*\*\*\*\*');

console.log(res);

console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

// Метод findOne - позволяет найти одну запись в коллекции с указанными параметрами

collection.findOne({age : {$lt : 30} }).then(function(value){

console.log('\*\*\*\*\*\*\*\*\*\* Employee with age less than 30 \*\*\*\*\*\*\*\*\*\*');

console.log(value);

console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

// закрываем подключение к базе данных

db.close();

});

});

});

});

### delete

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost:27017/userCollectionDB';

// Для удаления объектов (документов) из коллекции используется используется несколько методов

// deleteMany() - удаляет все документы, которые соответствуют определенному критерию

// deleteOne() - удаляет один документ, который соответствует определенному критерию

// findOneAndDelete() - получает и удаляет один документ, который соответствует определенному критерию

// drop() - удаляет всю коллекцию

MongoClient.connect(url, function(err, db){

var collection = db.collection('users');

collection.find().toArray(function(err, res){

console.log(res);

});

// collection.deleteOne({name : "Sergey", age: 37}, function(err, result){

// console.log(result);

// collection.find().toArray(function(err, res){

// console.log(res);

// db.close();

// });

// });

collection.deleteMany({name: "Ivan"}, function(err, result){

console.log(result);

db.close();

});

});

### update

// Для удаления объектов (документов) из коллекции используется используется несколько методов

// updateOne: обновляет один документ, который соответствует критерию фильтрации, и возвращает информацию об операции обновления

// updateMany: обновляет все документы, которые соответствуют критерию фильтрации, и возвращает информацию об операции обновления

// findOneAndUpdate: обновляет один документ, который соответствует критерию фильтрации, и возвращает обновленный документ

var mongoClient = require("mongodb").MongoClient;

var users = [{name: "Bob", age: 34} , {name: "Alice", age: 21}, {name: "Tom", age: 45}];

mongoClient.connect("mongodb://localhost:27017/userCollectionDB", function(err, db){

var collection = db.collection("users");

collection.insertMany(users, function(err, results){

collection.findOneAndUpdate(

// критерий выборки

{age: 25},

// параметр обновления

{ $set: {age: 21}},

// доп. опции обновления

{

returnOriginal: false

},

function(err, result){

console.log(result);

db.close();

}

);

});

});

## MSSQL

### connect

var express = require('express');

var app = express();

var port = 8080;

var mssql = require('mssql');

// параметры соединения с бд

var config = {

    user: 'test',                 // пользователь базы данных

    password: '12345',              // пароль пользователя

    server: 'localhost',            // хост

    database: 'testdb',          // имя бд

    port: 1433,                     // порт, на котором запущен sql server

pool: {

max: 10,                    // максимальное допустимое количество соединений пула

min: 0,                    // минимальное допустимое количество соединений пула

idleTimeoutMillis: 30000    // время ожидания перед завершением неиспользуемого соединения

}

}

app.use(function(req, res) {

    // mssql.ConnectionPool -

    var connection = new mssql.ConnectionPool(config);

    connection.connect(function(err){

        // mssql.Request -

        var request = new mssql.Request(connection);

        // request.query -

        request.query('SELECT \* FROM items', function(err, data) {

            if (err){

                console.log(err);

                return;

            }

            else {

                console.log(data);

                res.send(data.recordset);

            }

        });

    });

});

app.listen(port, function() {

    console.log('app listening on port ' + port);

});

### data records

var express = require('express');

var app = express();

var port = 8080;

var mssql = require('mssql');

// параметры соединения с бд

var config = {

    user: 'test',                 // пользователь базы данных

    password: '12345',              // пароль пользователя

    server: 'localhost',            // хост

    database: 'testdb',          // имя бд

    port: 1433,                     // порт, на котором запущен sql server

    pool: {

        max: 10,                    // максимальное допустимое количество соединений пула

        min: 0,                    // минимальное допустимое количество соединений пула

        idleTimeoutMillis: 30000    // время ожидания перед завершением неиспользуемого соединения

    }

}

app.use(function (req, res) {

    var connection = new mssql.ConnectionPool(config);

    connection.connect(function (err) {

        // Для выполнения запросов к бд используется метод request.query(command, callback(err, data))

        // метод query принимает такие аргументы:

        // command - выражение t-sql

        // callback - функция с параметрами err(ошибка) и data(результат запроса к бд)

        var request = new mssql.Request(connection);

        request.query('SELECT \* FROM items', function (err, data) {

            if (err) console.log(err);

            else {

                var html = ``;

                var allItems = data.recordset;

                for (let i = 0; i < allItems.length; i++) {

                    html += `<h3> ${allItems[i].name} - ${allItems[i].description}</h3>`

                }

                res.send(html);

                // завершить соединение

                connection.close();

            }

        });

    });

});

app.listen(port, function () {

    console.log('app listening on port ' + port);

});

### query stream

var express = require('express');

var app = express();

var port = 8080;

var mssql = require('mssql');

// параметры соединения с бд

var config = {

    user: 'test',                 // пользователь базы данных

    password: '12345',              // пароль пользователя

    server: 'localhost',            // хост

    database: 'testdb',          // имя бд

    port: 1433,                     // порт, на котором запущен sql server

pool: {

max: 10,                    // максимальное допустимое количество соединений пула

min: 0,                    // минимальное допустимое количество соединений пула

idleTimeoutMillis: 30000    // время ожидания перед завершением неиспользуемого соединения

}

};

app.use(function(req, res) {

    mssql.connect(config, function(err) {

        var html = ``;

        // для постепенной обработки данных по мере их поступления в node-mssql используется метод query и обработчики событий

        var request = new mssql.Request();

        request.stream = true; // включить режим потока данных

        request.query('select \* from items');

        // событие обработки колонки таблицы бд

        request.on('recordset', function(columns) {

            console.log(columns);

        });

        // событие обработки ряда таблицы бд

        request.on('row', function(row) {

            html += `<h2>${row.id} ${row.name} ${row.description}</h2>`;

        });

        // обработчик ошибок

        request.on('error', function(err) {

            console.log(err);

        });

        // обработчик события завершения запроса

        request.on('done', function(affected) {

            res.send(html);

            console.log('done');

        });

    });

});

app.listen(port, function() {

    console.log('app listening on port ' + port);

});

### query param

var express = require('express');

var app = express();

var port = 8080;

var mssql = require('mssql');

// параметры соединения с бд

var config = {

    user: 'test',                 // пользователь базы данных

    password: '12345',              // пароль пользователя

    server: 'localhost',            // хост

    database: 'testdb',          // имя бд

    port: 1433,                     // порт, на котором запущен sql server

pool: {

max: 10,                    // максимальное допустимое количество соединений пула

min: 0,                    // минимальное допустимое количество соединений пула

idleTimeoutMillis: 30000    // время ожидания перед завершением неиспользуемого соединения

}

};

app.use(function(req, res) {

    var connection = new mssql.ConnectionPool(config);

    connection.connect(function(err){

         // конструктор для предварительного форматирования запросов к бд - PreparedStatement

            var ps = new mssql.PreparedStatement(connection);

            // метод input позволяет указать значение параметра, который будет использован в запросе

            // аргументы:

            // name - имя параметра

            // type - SQL тип данных

            ps.input('param', mssql.Int);

            // подготовка запроса

            ps.prepare('SELECT \* FROM items WHERE id=@param ', function(err) {

                if (err) console.log(err);

                // выполнение запроса

                ps.execute({param: 2}, function(err, data) {

                    if (err) console.log(err);

                    res.send(data.recordset);

                    console.log('prepared statement executed');

                });

            });

    });

});

app.listen(port, function() {

    console.log('app listening on port ' + port);

});

### transaction

var express = require('express');

var mssql = require('mssql');

var app = express();

var port = 8080;

// параметры соединения с бд

var config = {

    user: 'test',                 // пользователь базы данных

    password: '12345',              // пароль пользователя

    server: 'localhost',            // хост

    database: 'testdb',          // имя бд

    port: 1433,                     // порт, на котором запущен sql server

    pool: {

        max: 10,                    // максимальное допустимое количество соединений пула

        min: 0,                    // минимальное допустимое количество соединений пула

        idleTimeoutMillis: 30000    // время ожидания перед завершением неиспользуемого соединения

    }

};

var connection = new mssql.ConnectionPool(config);

app.get('/', function (req, res) {

    connection.connect(function (err) {

        // транзакция - безопасная операция над бд с возможностью отката изменений в случае ошибки при выполнении запроса

        var transaction = new mssql.Transaction(connection);

        transaction.begin(function (err) {

            var request = new mssql.Request(transaction);

            request.query("INSERT INTO items (name, description) VALUES ('test item', 'some text')", function (err, data) {

                if (err) {

                    console.log(err);

                    transaction.rollback(function (err) {

                        console.log('rollback successful');

                    });

                } else {

                    transaction.commit(function (err, data) {

                            console.log('data commit success');

                            res.send('transaction successful');

                    });

                };

            });

        });

    });

});

// демонстрация отката изменений в случае ошибки при выполнении запроса к бд

app.get('/error', function (req, res) {

    var transaction = new mssql.Transaction(connection);

    transaction.begin(function (err) {

        var request = new mssql.Request(transaction);

        request.query("bad sql", function (err, data) {

            if (err) {

                console.log(err);

                transaction.rollback(function (err) {

                    if (err) {

                        console.log('rollback error');

                    }

                    else {

                        console.log('rollback successful');

                        res.send('transaction rollback successful');

                    }

                });

            } else {

                transaction.commit(function (err, data) {

                    if (err) {

                        console.log('could not commit data');

                    }

                    else {

                        console.log('data commit success');

                        res.send('transaction successful');

                    };

                });

            };

        });

    });

});

app.listen(port, function () {

    console.log('app listening on port ' + port);

});

# Cookies

## Cookie

var express = require('express');

var app = express();

app.use(function(req, res){

console.log(req.headers['cookie']);

// Отправляем файл

res.sendFile(\_\_dirname + '/index.html');

});

app.listen(8080, function(){

console.log('Server started');

});

## Headers

var express = require('express');

var app = express();

app.use('/', function (req, res) {

console.log('Cookies from client:', req.headers['cookie']);

// Метод позволяет записать в заголовок только один кукис

//res.setHeader('Set-Cookie', 'TestHeader=HeaderValue');

// Метод принимает имя заголовка - первым параметром и его значение - вторым параметром

res.setHeader('Set-Cookie', ['item3=value3', 'item4=value4']);

// Метод позволяет задать статус код ответа и объект с заголовками

//res.writeHead(200, { 'Set-Cookie': ['item1=value1', 'item2=value2'] });

console.log('Method getCookie():', res.getHeader('Set-Cookie'));

// записываем в ответ содержиое файла

res.sendFile(\_\_dirname + '/index.html');

});

app.listen(8080, function () {

console.log('Server start on port', 8080);

});

## CookieParser

var express = require('express');

var cookieParser = require('cookie-parser');

var app = express();

app.use(cookieParser('Secret string'));

app.get('/', function(req, res){

// Установка куков

res.cookie('login', 'admin', { maxAge : 10000 });

console.log(req.cookies);

// Удаление куков

res.clearCookie('login');

res.end();

});

app.listen(8080);

// Documentation - https://www.npmjs.com/package/cookie-parser

## Session views

// Сеансы - один из способов хранения состояния. Для работы с сеансами в express установим модуль express-session

// При использовании этого middleware сессии будут автоматически создаваться для кажого пользователя и все сессионные переменные

// автоматически попадут в req.session

var cookieSession = require('cookie-session')

var express = require('express')

var app = express()

app.use(cookieSession({

// Имя сессии

name: 'session',

// серкретные ключи

keys: ['key1', 'key2']

}));

app.get('/', function (req, res, next) {

console.log(req.ip)

// Создаем свойство в сессии

if(req.session.isNew)

console.log('Session created!');

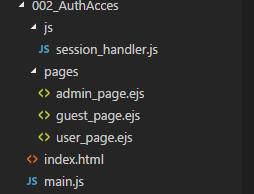
req.session.views = (req.session.views || 0) + 1;

res.end(req.session.views + ' views')

})

app.listen(8080)

# Authentication



<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet " />

<title>Log in</title>

<script>

window.onload = function() {

var btn1 = document.getElementById('btn1');

var inputs = document.getElementsByTagName('input');

btn1.addEventListener('click', function() {

var xhr = new XMLHttpRequest();

xhr.open('POST', '/login');

// create object with user data

var userData = {

username: inputs[0].value,

password: inputs[1].value

};

// Set the content type

xhr.setRequestHeader('Content-Type', 'application/json');

// send data

xhr.send(JSON.stringify(userData));

xhr.onload = function() {

alert(this.responseText);

};

xhr.onerror = function() {

alert('server error!');

}

});

            var btn2 = document.getElementById('btn2');

            btn2.onclick = function() {

                var xhr = new XMLHttpRequest();

                xhr.open('GET', '/logout');

                xhr.send();

                xhr.onload = function() {

                    alert(this.responseText);

                };

            };

            var btns = document.querySelector('.btns');

            btns.onclick = function(e) {

                if (!e.target.dataset.page) return false;

                var target = e.target.dataset.page;

                var xhr = new XMLHttpRequest();

                xhr.open('GET', '/' + target);

                xhr.send();

                xhr.onload = function() {

                    window.location.href="/" + target

                }

            }

}

</script>

</head>

<body>

<div class="panel well" style="width: 350px; padding: 25px; text-align: center">

<h2 style="margin-bottom: 35px">Log in</h2>

<form name="login">

<div class="form-group">

<label>Username</label> <input class="form-control input-lg" type="text" id="login" />

</div>

<div class="form-group">

<label>Password</label> <input class="form-control input-lg" type="text" id="password" />

</div>

<div class="form-group">

<input type="button" value="Log in" class="btn btn-lg btn-success" id="btn1" />

                <button value="Log out" class="btn btn-lg btn-danger" id="btn2">Log out</button>

</div>

</form>

</div>

    <div class="panel btns">

    <h2>Pages: </h2>

    <button data-page="admin" class="btn btn-info btn-lg">Admin page</button>

    <button data-page="user" class="btn btn-success btn-lg">User page</button>

    <button data-page="guest" class="btn btn-warning btn-lg">Guest page</button>

    </div>

</body>

</html>

var express = require('express');

var app = express();

var cookieParser = require('cookie-parser');

var session = require('express-session');

var bodyParser = require('body-parser');

var path = require('path');

var jsonParser = bodyParser.json();

app.use(jsonParser);

var port = 8080;

// зарегистрированные пользователи, которые могут быть авторизованы

var users = [

{ username: 'admin', password: '12345' },

{ username: 'foo', password: 'bar' },

{ username: 'user', password: 'test' }

]

// создание хранилища для сессий

//var sessionHandler = require('./js/session\_handler');

//var store = sessionHandler.createStore();

// создание сессии

app.use(cookieParser());

app.use(session({

//store: store,

resave: false,

saveUninitialized: true,

secret: 'supersecret'

}));

app.set('views', path.join(\_\_dirname, 'pages'));

app.set('view engine', 'ejs');

app.get('/', function (req, res) {

res.sendFile(path.join(\_\_dirname, 'index.html'));

});

app.post('/login', function (req, res) {

var foundUser;

// поиск пользователя в массиве users

for (var i = 0; i < users.length; i++) {

var u = users[i];

if (u.username == req.body.username && u.password == req.body.password) {

foundUser = u.username

};

};

if (foundUser !== undefined) {

req.session.username = req.body.username;

console.log("Login succeeded: ", req.session.username);

res.send('Login successful: ' + 'sessionID: ' + req.session.id + '; user: ' + req.session.username);

} else {

console.log("Login failed: ", req.body.username)

res.status(401).send('Login error');

}

});

app.get('/logout', function (req, res) {

req.session.username = '';

console.log('logged out');

res.send('logged out!');

});

// ограничение доступа к контенту на основе авторизации

app.get('/admin', function (req, res) {

// страница доступна только для админа

if (req.session.username == 'admin') {

console.log(req.session.username + ' requested admin page');

res.render('admin\_page');

} else {

res.status(403).send('Access Denied!');

}

});

app.get('/user', function (req, res) {

// страница доступна для любого залогиненного пользователя

if (req.session.username.length > 0) {

console.log(req.session.username + ' requested user page');

res.render('user\_page');

} else {

res.status(403).send('Access Denied!');

};

});

app.get('/guest', function (req, res) {

// страница без ограничения доступа

res.render('guest\_page');

});

app.listen(port, function () {

console.log('app running on port ' + port);

})

var cookieParser = require('cookie-parser');

var session = require('express-session');

// подключение модуля connect-mssql

var MSSQLStore = require('connect-mssql')(session);

var mssql = require('mssql');

module.exports = {

createStore: function () {

var config = {

user: 'test', // пользователь базы данных

password: '12345',   // пароль пользователя

server: 'localhost', // хост

database: 'testdb', // имя бд

port: 1433,          // порт, на котором запущен sql server

pool: {

max: 10, // максимальное допустимое количество соединений пула

min: 0, // минимальное допустимое количество соединений пула

idleTimeoutMillis: 30000 // время ожидания перед завершением неиспользуемого соединения

}

}

return new MSSQLStore(config);

}

}

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title>Admin Page</title>

</head>

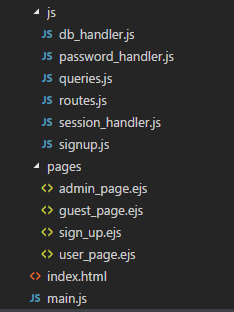
<body>

<h1>Hello, Admin!</h1>

</body>

</html>

## Crypt auth



var crypto = require('crypto');

module.exports = {

encrypt\_pass: function (password) {

var hash = crypto.createHmac('sha1', 'abc').update(password).digest('hex');

return hash;

}

}

var mssql = require('mssql');

var connection = require('./db\_handler');

var pass\_handler = require('./password\_handler');

module.exports = {

    curr\_user: '',

    get\_users: function(req, res){

        var request = new mssql.Request(connection);

        request.query('SELECT \* FROM users', function(err, rows) {

            var users = rows.map((row) => {

                return '<h3> ' + row.username + ' </h3>'

            });

            res.send(users.join(''));

        })

    },

    check\_user: function(req, res, next) {

        var self = this;

        var ps = new mssql.PreparedStatement(connection);

        var inserts = {

            username: req.body.username

        };

        ps.input('username', mssql.Text);

        ps.prepare('SELECT \* FROM users WHERE username LIKE @username', function(err) {

            if (err) console.log(err);

            ps.execute(inserts, function(err, rows) {

                if (err) console.log(err);

                ps.unprepare();

                // имя пользователя найдено

                if (rows.length > 0) {

                    self.curr\_user = rows[0].username;

                    // перейти к проверке пароля

                    next()

                }

                // имя пользователя не найдено

                else {

                    res.status(404).send('user not found!');

                }

            })

        });

    },

// проверка пароля

check\_pass: function (req, res) {

        var self = this;

        var inserts = {

            password\_hash: pass\_handler.encrypt\_pass(req.body.password) // хэширование пароля

        }

        var ps = new mssql.PreparedStatement(connection);

        ps.input('password\_hash', mssql.Text);

        ps.prepare('SELECT \* FROM users WHERE password LIKE @password\_hash', function(err) {

            if (err) console.log(err);

            ps.execute(inserts, function(err, rows) {

                if (err) console.log(err);

                ps.unprepare();

                // пароль верный

                if (rows) {

                    req.session.username = self.curr\_user;

                    res.status(200).send('user ' + req.session.username + ' logged in!');

                }

                // пароль неверный

                else {

                    res.status(404).send('wrong password!');

                }

            })

        });

    }

}

module.exports = function (app) {

    app.get('/logout', function(req, res) {

        req.session.username = '';

        console.log('logged out');

        res.send('logged out!');

    });

    // ограничение доступа к контенту на основе авторизации

    app.get('/admin', function (req, res) {

        // страница доступна только для админа

        if (req.session.username == 'admin') {

            console.log(req.session.username + ' requested admin page');

            res.render('admin\_page');

        } else {

            res.status(403).send('Access Denied!');

        }

    });

    app.get('/user', function (req, res) {

        // страница доступна для любого залогиненного пользователя

        if (req.session.username.length > 0) {

            console.log(req.session.username + ' requested user page');

            res.render('user\_page');

        } else {

            res.status(403).send('Access Denied!');

        }

    });

    app.get('/guest', function (req, res) {

        // страница без ограничения доступа

        res.render('guest\_page');

    })

}

var cookieParser = require('cookie-parser');

var session = require('express-session');

// подключение модуля connect-mssql

var MSSQLStore = require('connect-mssql')(session);

var mssql = require('mssql');

module.exports = {

createStore: function () {

var config = {

user: 'test', // пользователь базы данных

password: '12345',   // пароль пользователя

server: 'localhost', // хост

database: 'testdb', // имя бд

port: 1433,          // порт, на котором запущен sql server

pool: {

max: 10, // максимальное допустимое количество соединений пула

min: 0, // минимальное допустимое количество соединений пула

idleTimeoutMillis: 30000 // время ожидания перед завершением неиспользуемого соединения

}

};

return new MSSQLStore(config);

}

}

var mssql = require('mssql');

var connection = require('./db\_handler');

var pass\_handler = require('./password\_handler');

module.exports = {

    addUser: function(req, res) {

        var inserts = {

            username: req.body.username,

            password\_hash: pass\_handler.encrypt\_pass(req.body.password)

        };

        var ps = new mssql.PreparedStatement(connection);

        ps.input('username', mssql.Text);

        ps.input('password\_hash', mssql.Text);

        ps.prepare('INSERT INTO users (username, password) VALUES (@username, @password\_hash)', function(err) {

            if (err) console.log(err);

            ps.execute(inserts, function(err, rows) {

                if (!err) res.status(200).send('user created successfully!');

                else console.log(err);

                ps.unprepare();

            })

        })

    }

}

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet " />

<title>Log in</title>

<script>

window.onload = function() {

            var xhr = new XMLHttpRequest();

            xhr.open('GET', '/all');

            xhr.send();

            xhr.onload = function() {

                document.getElementById('container').innerHTML = this.responseText;

            }

var btn = document.getElementById('btn');

var inputs = document.getElementsByTagName('input');

btn.addEventListener('click', function() {

var xhr = new XMLHttpRequest();

xhr.open('POST', '/login');

var userData = {

username: inputs[0].value,

password: inputs[1].value

};

xhr.setRequestHeader('Content-Type', 'application/json');

xhr.send(JSON.stringify(userData));

xhr.onload = function() {

alert(this.responseText);

};

xhr.onerror = function() {

alert('server error!');

}

})

            var btn2 = document.getElementById('btn2');

            btn2.onclick = function() {

                var xhr = new XMLHttpRequest();

                xhr.open('GET', '/logout');

                xhr.send();

                xhr.onload = function() {

                    alert(this.responseText);

                }

            }

            var btns = document.querySelector('.btns');

            btns.onclick = function(e) {

                if (!e.target.dataset.page) return false;

                var target = e.target.dataset.page;

                var xhr = new XMLHttpRequest();

                xhr.open('GET', '/' + target);

                xhr.send();

                xhr.onload = function() {

                    window.location.href="/" + target

                }

            }

}

</script>

</head>

<body>

<div class="panel well" style="width: 350px; padding: 25px; text-align: center">

<h2 style="margin-bottom: 35px">Log in</h2>

<form name="login">

<div class="form-group">

<label>Username</label> <input class="form-control input-lg" type="text" id="login" />

</div>

<div class="form-group">

<label>Password</label> <input class="form-control input-lg" type="text" id="password" />

</div>

<div class="form-group">

<input type="button" value="Log in" class="btn btn-lg btn-success" id="btn" />

                <button value="Log out" class="btn btn-lg btn-danger" id="btn2">Log out</button>

</div>

</form>

</div>

    <div class="panel">

    <h1>Registered users: </h1>

    <div id="container"></div>

    </div>

    <div class="panel btns">

    <h2>Pages: </h2>

    <button data-page="admin" class="btn btn-info btn-lg">Admin page</button>

    <button data-page="user" class="btn btn-success btn-lg">User page</button>

    <button data-page="guest" class="btn btn-warning btn-lg">Guest page</button>

    </div>

</body>

</html>

var express = require('express');

var app = express();

var cookieParser = require('cookie-parser');

var session = require('express-session');

var bodyParser = require('body-parser');

var path = require('path');

var jsonParser = bodyParser.json();

app.use(jsonParser);

var port = 8080;

// создание хранилища для сессий

var sessionHandler = require('./js/session\_handler');

var store = sessionHandler.createStore();

// создание сессии

app.use(cookieParser());

app.use(session({

store: store,

resave: false,

saveUninitialized: true,

secret: 'supersecret'

}));

var handlers = require('./js/queries');

var signup = require('./js/signup');

var routes\_hangler = require('./js/routes')(app);

app.set('views', path.join(\_\_dirname, 'pages'));

app.set('view engine', 'ejs');

app.get('/', function (req, res) {

res.render('sign\_up');

});

app.get('/login', function (req, res) {

res.sendFile(path.join(\_\_dirname, 'index.html'));

});

app.get('/all', handlers.get\_users);

app.post('/login', handlers.check\_user);

app.post('/login', handlers.check\_pass);

// регистрация пользователя

app.post('/signup', signup.addUser);

// ограничение доступа к контенту на основе авторизации

app.get('/check', function (req, res) {

if (req.session.username) {

res.send('hello, user ' + req.session.username);

} else {

res.send('Not logged in(');

}

});

app.listen(port, function () {

console.log('app running on port ' + port);

})

## Multer

var express = require('express');

var app = express();

var path = require('path');

var port = 8080;

// multer - middleware для обработки данных формы в кодировке multipart/form-data

var multer = require('multer');

var upload = multer();

app.get('/', function(req, res) {

    res.sendFile(path.join(\_\_dirname, 'index.html'));

})

// конфигурация модуля multer

var form\_data = upload.fields([{name: 'username'}, {name: 'email'}]);

app.post('/form\_handler', form\_data, function(req, res) {

    // при использовании модуля multer данные полей формы доступны через объект req.body

    console.log(req.body);

    res.send(`<h2>user data:</h2> <p>username: ${req.body.username}</p> <p>email: ${req.body.email}</p>`);

})

app.listen(port, function() {

     console.log('App listening on port ' + port);

});

## FileUpload

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title>File upload demo</title>

</head>

<body>

<form action="/upload" method="POST" enctype="multipart/form-data">

<input type="file" name="recfile" />

<input type="submit">

</form>

</body>

</html>

var express = require('express');

var port = 8080;

var path = require('path');

var fs = require('fs');

// multer - middleware для обработки данных формы в кодировке multipart/form-data

var multer = require('multer');

// dest - директория для сохранения файлов, загружаемых на сервер

var upload = multer({dest: \_\_dirname + '/uploads'});

// указать, что будет загружен один файл с именем recfile.

// имя файла может быть любым, но оно должно совпадать со значением атрибута name элемента формы input с типом file

// например, <input type="file" name="recfile" />

var type = upload.single('recfile');

var app = express();

app.get('/', function(req,res, next) {

     res.sendFile(\_\_dirname + '/index.html');

})

// указать файл при обработке POST запроса (второй аргумент метода app.post)

app.post('/upload', type, function(req, res) {

     // загруженный файл доступен через свойство req.file

     console.log(req.file);

     // файл временного хранения данных

     var tmp\_path = req.file.path;

     // место, куда файл будет загружен

     var target\_path = path.join(req.file.destination, req.file.originalname);

     // загрузка файла

     var src = fs.createReadStream(tmp\_path);

     var dest = fs.createWriteStream(target\_path);

     src.pipe(dest);

     // обработка результатов загрузки

     src.on('end', function() {

        // удалить файл временного хранения данных

        fs.unlink(tmp\_path);

        res.send('complete');

     });

     src.on('error', function(err) {

       // удалить файл временного хранения данных

     fs.unlink(tmp\_path);

        res.send('error');

     });

})

app.listen(port, function() {

     console.log('App listening on port ' + port);

});

# Rest API

## Route methods



var express = require('express');

var app = express();

var port = 8080;

var router = require('./route.js')

// использовать роутер на пути /api

app.use('/api', router.rout);

app.listen(port, function () {

console.log('app running on port ' + port);

});

var express = require('express');

var router = express.Router();

// В данном приложении показана базовая логика построения RESTful API

router.get('/:id', function (req, res) {

res.status(200).send('get item by ID ' + req.params.id + '!');

});

router.post('/', function (req, res) {

res.status(201).send('create item!');

});

router.put('/:id', function (req, res) {

res.status(200).send('update item by ID ' + req.params.id + '!');

});

router.delete('/:id', function (req, res) {

res.status(200).send('remove item by ID ' + req.params.id + '!');

});

module.exports = { rout: router };

## CORS

var express = require('express');

var app = express();

var router = express.Router();

var port = 8080;

var router = require('./route')

// middleware для использования CORS

app.use(function (req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

// использовать роутер на пути /api

app.use('/api', router.rout);

app.listen(port, function () {

console.log('app running on port ' + port);

});

var express = require('express');

var router = express.Router();

router.get('/:id', function (req, res) {

res.status(200).send('get item by ID ' + req.params.id + '!');

});

router.post('/', function (req, res) {

res.status(201).send('create item!');

});

router.put('/:id', function (req, res) {

res.status(200).send('update item by ID ' + req.params.id + '!');

});

router.delete('/:id', function (req, res) {

res.status(200).send('remove item by ID ' + req.params.id + '!');

});

module.exports = { rout : router };

## CORS DB

var express = require('express');

var app = express();

var router = express.Router();

var port = 8080;

var mssql = require('mssql');

var config = {

user : 'test',

password : '12345',

database : 'testdb',

server : 'localhost',

port : 1433,

pool: {

max: 10, // максимальное допустимое количество соединений пула

min: 0, // минимальное допустимое количество соединений пула

idleTimeoutMillis: 30000 // время ожидания перед завершением неиспользуемого соединения

}

}

// ConnectionPool - создает объект который позволяет подключаться к БД

var connection = new mssql.ConnectionPool(config);

var pool = connection.connect(function(err) {

    if (err) console.log(err)

});

// middleware для использования CORS

app.use(function (req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

router.get('/', function (req, res) {

// подключение к бд

    var request = new mssql.Request(connection);

    request.query("SELECT \* FROM items", function(err, rows) {

        if (err) console.log(err);

console.log('GET ' + req.url);

        res.status(200).send('selected items: ' + JSON.stringify(rows.recordset));

    });

});

router.get('/:id', function (req, res) {

// подключение к бд

    var ps = new mssql.PreparedStatement(connection);

    var inserts = {

        id: parseInt(req.params.id)

    }

    ps.input('id', mssql.Int);

    ps.prepare('SELECT \* FROM items WHERE id=@id', function(err) {

        if (err) console.log(err);

        ps.execute(inserts, function(err, rows) {

                if (err) console.log(err);

                console.log('GET ' + req.url);

                res.status(200).send('selected item: ' + JSON.stringify(rows.recordsets));

                ps.unprepare();

        });

    });

});

router.post('/', function (req, res) {

// подключение к бд

    var request = new mssql.Request(connection);

    request.query("INSERT INTO items (name, description, completed) VALUES ('Test', 'Some text', 1)", function(err, rows) {

        if (err) console.log(err);

console.log('POST ' + req.url);

res.status(200).send('sample item added to database');

    });

});

router.put('/:id', function (req, res) {

// подключение к бд

    var ps = new mssql.PreparedStatement(connection);

    var inserts = {

        id: parseInt(req.params.id)

    }

    ps.input('id', mssql.Int);

    ps.prepare("UPDATE items SET name='new Name', description='Some other text', completed=0 WHERE id=@id", function(err) {

        if (err) console.log(err);

        ps.execute(inserts, function(err, rows) {

            if (err) console.log(err);

            console.log('PUT ' + req.url);

res.status(200).send('item by id ' + req.params.id + ' changed');

            ps.unprepare();

        });

    });

});

router.delete('/:id', function (req, res) {

// подключение к бд

    var ps = new mssql.PreparedStatement(connection);

    var inserts = {

        id: parseInt(req.params.id)

    }

    ps.input('id', mssql.Int);

    ps.prepare('DELETE FROM items WHERE id=@id', function(err) {

        if (err) console.log(err);

        ps.execute(inserts, function(err, rows) {

            if (err) console.log(err);

console.log('DELETE ' + req.url);

res.status(200).send('item deleted from database');

            ps.unprepare();

        });

    });

});

// использовать роутер на пути /api

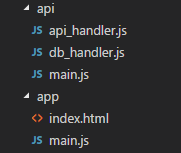
app.use('/api', router);

app.listen(port, function () {

console.log('app running on port ' + port);

});

CORS API and APP



/ соединение с бд

var connection = require('./db\_handler');

var mssql = require('mssql');

var path = require('path');

module.exports = {

// загрузка всех элементов

loadItems: function (req, res) {

        // подключение к бд

        var request = new mssql.Request(connection);

        request.query("SELECT \* FROM items", function(err, rows) {

            if (err) console.log(err);

            console.log('GET ' + req.url);

            res.json(rows);

        });

},

// загрузка элемента из бд по id

getItemById: function (req, res) {

        // подключение к бд

        var ps = new mssql.PreparedStatement(connection);

        var inserts = {

            id: parseInt(req.params.id)

        }

        ps.input('id', mssql.Int);

        ps.prepare('SELECT \* FROM items WHERE id=@id', function(err) {

            if (err) console.log(err);

            ps.execute(inserts, function(err, rows) {

                    if (err) console.log(err);

                    console.log('GET ' + req.url);

                    res.json(rows['recordset'][0]);

                    ps.unprepare();

            });

        });

},

// создание элемента

createItem: function (req, res) {

// подключение к бд

        var ps = new mssql.PreparedStatement(connection);

        var data = req.body;

        var \_id = Math.ceil(Math.random() \* 100);

        var inserts = {

            id : \_id,

            name: data.name,

            description: data.description,

            completed: parseInt(data.completed)

        }

        ps.input('id', mssql.Int);

        ps.input('name', mssql.Text);

        ps.input('description', mssql.Text);

        ps.input('completed', mssql.Int);

        ps.prepare("INSERT INTO items (id, name, description, completed) VALUES (@id, @name, @description, @completed)", function(err) {

            if (err) console.log(err);

            ps.execute(inserts, function(err, rows) {

console.log('item created');

res.status(201).send('item created');

                 ps.unprepare();

            });

        });

},

// обновление элемента (редактирование)

updateItem: function (req, res) {

        var ps = new mssql.PreparedStatement(connection);

        var data = req.body;

        var inserts = {

            name: data.name,

            description: data.description,

            completed: parseInt(data.completed),

            id: parseInt(data.id)

        }

        ps.input('name', mssql.Text);

        ps.input('description', mssql.Text);

        ps.input('completed', mssql.Int);

        ps.input('id', mssql.Int);

        ps.prepare('UPDATE items SET name=@name, description=@description, completed=@completed WHERE id=@id', function(err) {

            if (err) console.log(err);

            ps.execute(inserts, function(err, rows) {

                if (err) console.log(err);

                console.log('PUT ' + req.url);

res.status(200).send('item updated');

                ps.unprepare();

            });

        });

},

// удаление элемента

removeItem: function (req, res) {

        var ps = new mssql.PreparedStatement(connection);

        var inserts = {

            id: parseInt(req.params.id)

        }

        ps.input('id', mssql.Int);

        ps.prepare('DELETE FROM items WHERE id=@id', function(err) {

            if (err) console.log(err);

            ps.execute(inserts, function(err, rows) {

                if (err) console.log(err);

                console.log('DELETE ' + req.url);

res.status(200).send('item deleted');

                ps.unprepare();

            });

        });

}

}

// Подключаем модуль mssql для работы с mssql сервером

var mssql = require('mssql');

var config = {

user : 'test',                      // пользователь базы данных

password : '12345',                 // пароль пользователя

database : 'testdb',                // имя бд

server : 'localhost',               // хост

port : 1433,                        // порт, на котором запущен sql server

pool: {

max: 10,                        // максимальное допустимое количество соединений пула

min: 0,                        // минимальное допустимое количество соединений пула

idleTimeoutMillis: 30000        // время ожидания перед завершением неиспользуемого соединения

}

}

var connection = new mssql.ConnectionPool(config);

var pool = connection.connect(function(err) {

    if (err) console.log(err)

});

module.exports = pool;

var express = require('express');

var app = express();

var path = require('path');

var port = 8080;

var bodyParser = require('body-parser');

// подключение модуля для обработки запросов

var apiHandler = require('./api\_handler');

// middleware для обработки данных запросов в формате JSON

app.use(bodyParser.json());

// middleware для использования CORS

app.use(function (req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

    res.header("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE, OPTIONS");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

var router = express.Router();

// загрузка всех элементов из бд

router.get('/', apiHandler.loadItems)

// выбор элемента

router.get('/:id', apiHandler.getItemById);

// создание элемента

router.post('/new', apiHandler.createItem);

// обновление элемента (редактирование)

router.put('/:id', apiHandler.updateItem);

// удаление элемента

router.delete('/:id', apiHandler.removeItem);

app.use('/api', router);

app.listen(port, function () {

console.log('app running on port ' + port);

})

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />

<title>RESTful API demo</title>

<script src="https://code.jquery.com/jquery-3.1.0.min.js" integrity="sha256-cCueBR6CsyA4/9szpPfrX3s49M9vUU5BgtiJj06wt/s=" crossorigin="anonymous"></script>

<script>

$(document).ready(function () {

var tbody = document.getElementById('tbody');

// запрос для загрузки всех элементов

function getItems() {

$.ajax({

url: 'http://localhost:8080/api/',

accepts: 'application/json',

method: 'GET',

success: function (data) {

                        var items = data["recordsets"][0];

var html = '';

// форматирование данных ответа сервера

for (var i = 0; i < items.length; i++) {

html += `<tr>

<td class="itemID">${items[i].id}</td>

<td>${items[i].name}</td>

<td>${items[i].description}</td>

<td>${items[i].completed==1 ? 'yes' : 'no'}</td>

</tr>`

};

tbody.innerHTML = html;

}

});

}

getItems();

function validateID(id) {

var idIsValid = false;

for (var i = 0; i < $('.itemID').length; i++) {

if (id == $('.itemID').eq(i).text()) {

idIsValid = true;

}

}

console.log(idIsValid)

return idIsValid;

}

function checkIDUnique(id) {

var idIsUnique = true;

for (var i = 0; i < $('.itemID').length; i++) {

if (id == $('.itemID').eq(i).text()) {

idIsUnique = false;

}

}

return idIsUnique;

}

// запрос для просмотра элемента

$('#select').on('click', function () {

var itemID = $('#itemID1').val();

if (validateID(itemID)) {

$.ajax({

url: 'http://localhost:8080/api/' + itemID,

accept: 'application/json',

type: 'GET',

success: function (data) {

                            var item = data;

var html =

`<td>${item.id}</td>

<td>${item.name}</td>

<td>${item.description}</td>

<td>${item.completed==1 ? 'yes' : 'no'}</td> `

tbody.innerHTML = html;

$('#back').css({

display: 'block'

})

}

})

} else { alert('invalid ID!') }

})

// запрос для удаления элемента

$('#delete').on('click', function () {

var itemID = $('#itemID1').val();

if (validateID(itemID)) {

$.ajax({

url: 'http://localhost:8080/api/' + itemID,

accept: 'application/json',

type: 'DELETE',

success: function () {

window.location.href = '/'

}

})

} else { alert('invalid ID!') }

})

// запрос для обновления элемента

$('#update').on('click', function () {

var itemID = $('#itemID2').val();

console.log(itemID)

if (validateID(itemID)) {

if ($('#name').val() == '' || $('#description').val() == '') {

alert('empty field!');

return;

}

var requestData = {

id: $('#itemID2').val(),

name: $('#name').val(),

description: $('#description').val(),

get completed() {

return $('#completed').is(':checked') ? 1 : 0

}

}

$.ajax({

url: 'http://localhost:8080/api/' + itemID,

type: 'PUT',

data: JSON.stringify(requestData),

contentType: 'application/json',

success: function() {

window.location.href = '/'

}

})

} else { alert('invalid ID!') }

})

// запрос для создания элемента

$('#create').on('click', function () {

var itemID = $('#itemID2').val();

if ($('#name').val() == '' || $('#description').val() == '') {

alert('empty field!');

return;

}

var requestData = {

name: $('#name').val(),

description: $('#description').val(),

get completed() {

return $('#completed').is(':checked') ? 1 : 0

}

}

console.log(JSON.stringify(requestData))

console.log(requestData)

$.ajax({

url: 'http://localhost:8080/api/new' ,

type: 'POST',

contentType: 'application/json',

data: JSON.stringify(requestData),

success: function () {

window.location.href = '/'

}

})

})

})

</script>

</head>

<body>

<div class="panel well">

<table class="table table-striped">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Description</th>

<th>Completed</th>

</tr>

</thead>

<tbody id="tbody">

</tbody>

</table>

</div>

<div class="panel" style="padding-left: 25px; padding-bottom: 25px;">

<h3>View or delete item</h3>

<div class="form-group" style="width: 300px">

<a id="back" href="/" style="display: none"><h4>Back to items</h4></a>

<label>Item ID</label> <input id="itemID1" class="form-control input-lg">

</div>

<button id="select" class="btn btn-info btn-lg">View item by ID</button>

<button id="delete" class="btn btn-danger btn-lg">Delete item by ID</button>

</div>

<div class="panel" style="padding-left: 25px; padding-bottom: 25px;">

<h3>Create or edit item</h3>

<div class="form-group" style="width: 300px">

<div class="form-group" style="width: 300px">

<label>Item ID (only for updating items)</label> <input id="itemID2" class="form-control input-lg">

</div>

<label>Item Name</label> <input id="name" class="form-control input-lg" />

</div>

<div class="form-group" style="width: 300px">

<label>Item Description</label> <input id="description" class="form-control input-lg" />

</div>

<div class="form-group" style="width: 300px">

<label>Completed</label> <input id="completed" type="checkbox" />

</div>

<button id="update" class="btn btn-warning btn-lg">Update item by ID</button>

<button id="create" class="btn btn-success btn-lg">Create Item</button>

</div>

</body>

</html>

var express = require('express');

var app = express();

var http = require('http');

var path = require('path');

var bodyParser = require('body-parser');

app.use(bodyParser.json());

var port = 8081;

var portApi = 8080;

app.get('/', function (req, res) {

res.sendFile(path.join(\_\_dirname, 'index.html'));

});

var cb = function (response, res) {

response.on('data', function (chunk) {

res.write(chunk.toString());

});

response.on('end', function () {

res.end();

})

};

app.get('/all', function (req, res) {

var options = {

host: 'localhost',

port: portApi,

method: 'GET',

path: '/api/'

}

var request = http.request(options, function (response) {

cb(response)

});

request.end();

});

app.get('/view/:id', function (req, res) {

var options = {

host: 'localhost',

port: portApi,

method: 'GET',

path: '/api/' + req.params.id

}

var request = http.request(options, function (response) {

cb(response)

});

request.end();

});

app.post('/new', function (req, res) {

var options = {

host: 'localhost',

port: portApi,

method: 'POST',

        path: '/api/new' ,

        headers: {

'Content-Type': 'application/json'

}

}

var request = http.request(options, function (response) {

cb(response)

});

request.write(JSON.stringify(req.body));

request.end();

});

app.post('/edit/:id', function (req, res) {

var options = {

host: 'localhost',

port: portApi,

method: 'PUT',

path: '/api/' + req.params.id,

headers: {

'Content-Type': 'application/json'

}

}

var request = http.request(options, function (response) {

cb(response)

});

console.log('POST BODY:', req.body);

request.write(JSON.stringify(req.body));

request.end();

});

app.get('/delete/:id', function (req, res) {

var options = {

host: 'localhost',

port: 1337,

method: 'DELETE',

path: '/api/' + req.params.id

}

var request = http.request(options, function (response) {

cb(response, res)

});

request.end();

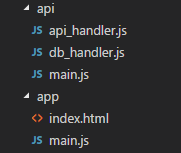
});

app.listen(port, function() {

    console.log('app running on port ' + port);

})

# Socket



// подключение express и socket.io

var app = require('express')();

var server = require('http').Server(app);

var io = require('socket.io')(server);

var path = require('path');

var port = 8080;

app.get('/', function (req, res) {

res.sendFile(path.join(\_\_dirname, 'index.html'));

});

io.on('connection', function (socket) {

// метод send автоматически генерирует событие 'message'

socket.send('Hello world')

// обработка события

socket.on('greeting', function (data) {

console.log(data);

});

})

server.listen(port, function () {

console.log('app running on port ' + port);

})

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title>Demo</title>

<script src="/socket.io/socket.io.js"></script>

<script>

// подклчение socket.io клиента к серверу

var socket = io.connect('http://localhost:8080');

// обработка события message

socket.on('message', function (data) {

console.log(data);

document.write(data);

// генерация события

socket.emit('greeting', { response: 'data received!' });

});

</script>

</head>

<body>

</body>

</html>

## Chat

window.onload = function () {

var btn = document.getElementById('btn');

var message\_input = document.getElementById('inp');

var message\_container = document.getElementById('messages');

var socket = io.connect('http://localhost:8080');

socket.on('chat message', function (message) {

console.log(message)

// сгенерировать html разметку сообщения

var display\_message = `<div class ="panel well">

<h4>Message: </h4>

<h5>${message.text}</h5>

</div>`

// добавить результат на страницу

message\_container.innerHTML += display\_message;

})

btn.onclick = function () {

// сгенерировать событие отправки сообщения

socket.emit('send message', { text: message\_input.value });

}

}

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title>Demo Chat</title>

<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />

<style>

body {

background-color: rgb(212, 219, 226);

overflow-x: hidden;

}

#msg\_form {

position: absolute;

margin-bottom: 0;

bottom: 0;

}

h1 {

text-align: center;

}

</style>

<script src="/socket.io/socket.io.js"></script>

<script src="client.js"></script>

</head>

<body>

<div id="wrap">

<div class="row">

<div class="col-md-12 panel">

<h1>Messages</h1>

</div>

</div>

<div class="row">

<div class="col-md-12">

<div id="messages"></div>

</div>

</div>

<div class="row well" id="msg\_form">

<div class="col-md-12">

<div class="input-group">

<input type="text" id="inp" class="form-control input-lg" placeholder="Message..." />

<span class="input-group-btn">

<button class="btn btn-lg btn-success" type="button" id="btn">Send!</button>

</span>

</div>

</div>

</div>

</div>

</body>

</html>

// подключение express и socket.io

var app = require('express')();

var server = require('http').Server(app);

var io = require('socket.io')(server);

var path = require('path');

var port = 8080;

// массив для хранения текущих подключений

var connections = [];

app.get('/', function (req, res) {

res.sendFile(path.join(\_\_dirname, 'index.html'));

});

app.get('/client.js', function (req, res) {

res.sendFile(path.join(\_\_dirname, 'client.js'));

});

// установка соединения

io.on('connection', function (socket) {

connections.push(socket);

console.log('Connected: %s sockets connected', connections.length);

// окончание соединения

socket.on('disconnect', function (data) {

connections.splice(connections.indexOf(socket), 1);

console.log('Disconnected: %s sockets connected', connections.length);

})

socket.on('send message', function (data) {

// сгенерировать событие chat message и отправить его всем доступным подключениям

io.sockets.emit('chat message', data);

})

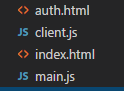
})

server.listen(port, function () {

console.log('app running on port ' + port);

})

## Chat



<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />

<script src="/socket.io/socket.io.js"></script>

<title>Enter Chat!</title>

<style>

body {

overflow: hidden;

background-color: rgb(212, 219, 226);

}

#username {

display: inline-block;

}

.col-md-12 {

text-align: center;

}

.wrap {

width: 500px;

display: inline-block;

margin-top: 350px;

}

</style>

<script>

window.onload = function () {

var btn = document.getElementsByTagName('button')[0];

var inp = document.getElementsByTagName('input')[0];

btn.onclick = function () {

if (inp.value == '') {

alert ('invalid username! try again!');

return;

}

// перенаправление на страницу чата

window.location.href = '/' + inp.value;

}

}

</script>

</head>

<body>

<div class="row">

<div class="col-md-12">

<div class="wrap">

<h1>Ignite chat!</h1>

<div class="input-group">

<input type="text" class="input-lg form-control" id="username" placeholder="Username..." />

<span class="input-group-btn">

<button class="btn btn-lg btn-warning">Connect</button>

</span>

</div>

</div>

</div>

</div>

</body>

</html>

var socket = io.connect('http://localhost:8080/');

var user = '';

window.onload = function () {

var users\_container = document.getElementById('userlist');

var message\_container = document.getElementById('messages');

message\_container.style.height = window.innerHeight - 200 + 'px';

var btn = document.getElementById('btn');

var message\_input = document.getElementById('inp');

// загрузить имена пользователей, которые online

socket.emit('load users');

socket.on('users loaded', function (data) {

var display\_users = data.users.map((username) => {

return `<li>${username}</li>`;

});

users\_container.innerHTML = display\_users.join(' ');

});

// загрузить сообщения других пользователей (при загрузке страницы)

socket.emit('load messages');

socket.on('messages loaded', function (data) {

var display\_messages = data.messages.map((msg) => {

return (`<div class ="panel well">

<h4>${msg.author}</h4>

<h5>${msg.text}</h5>

</div>`)

});

message\_container.innerHTML = display\_messages.join(' ');

});

// загрузить текущее сообщение

socket.on('chat message', function (message) {

console.log(message)

var display\_message = `<div class ="panel well">

<h4>${message.author}</h4>

<h5>${message.text}</h5>

</div>`

message\_container.innerHTML += display\_message;

});

// получить имя пользователя

socket.on('new user', function (data) {

user = data.name;

})

btn.onclick = function () {

// сгенерировать событие отправки сообщения

socket.emit('send message', { text: message\_input.value, author: user });

}

}

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title>Demo Chat</title>

<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />

<style>

body {

background-color: rgb(212, 219, 226);

overflow-x: hidden;

}

#msg\_form {

position: absolute;

margin-bottom: 0;

bottom: 0;

}

h1 {

text-align: center;

}

#users {

height: 300px;

padding-left: 30px;

opacity: 0.8;

}

#users ul {

list-style-type: none;

}

#messages {

overflow-y: scroll;

}

</style>

<script src="/socket.io/socket.io.js"></script>

<script src="client.js"></script>

</head>

<body>

<div id="wrap">

<div class="row">

<div class="col-md-12 panel">

<h1>Messages</h1>

</div>

</div>

<div class="row">

<div class="col-md-2 panel" id="users">

<h3>Users online: </h3>

<ul id="userlist"></ul>

</div>

<div class="col-md-10">

<div id="messages"></div>

</div>

</div>

<div class="row well" id="msg\_form">

<div class="col-md-12">

<div class="input-group">

<input type="text" id="inp" class="form-control input-lg" placeholder="Message..." />

<span class="input-group-btn">

<button class="btn btn-lg btn-success" type="button" id="btn">Send!</button>

</span>

</div>

</div>

</div>

</div>

</body>

</html>

// подключение express и socket.io

var app = require('express')();

var server = require('http').Server(app);

var io = require('socket.io')(server);

var path = require('path');

var port = 8080;

// массив для хранения текущих подключений

var connections = [];

// массив для хранения текущих пользователей

var users = [];

// массив для хранения текущих сообщений

var messages = [];

app.get('/', function (req, res) {

res.sendFile(path.join(\_\_dirname, 'auth.html'));

});

app.get('/:id', function (req, res) {

if (req.params.id == 'client.js') {

res.sendFile(path.join(\_\_dirname, 'client.js'));

}

else if (req.params.id == 'favicon.ico') {

res.sendStatus(404);

}

else {

users.push(req.params.id);

res.sendFile(path.join(\_\_dirname, 'index.html'));

}

})

// установка соединения

io.on('connection', function (socket) {

connections.push(socket);

console.log(users)

console.log('Connected: %s sockets connected', connections.length);

// окончание соединения

socket.on('disconnect', function (data) {

var index = connections.indexOf(socket)

// удалить разорванное соединение из списка текущих соединений

var deletedItem = connections.splice(index, 1);

// удалить пользователя из массива текущих пользователей

users.splice(index, 1);

// обновить список пользователей на клиенте

io.sockets.emit('users loaded', { users: users })

console.log('Disconnected: %s sockets connected', connections.length);

});

// обработка сообщения

socket.on('send message', function (data) {

// сохранить сообщение

messages.push(data);

// сгенерировать событие chat message и отправить его всем доступным подключениям

io.sockets.emit('chat message', data);

});

// загрузить пользователей

socket.on('load users', function () {

console.log(users)

io.sockets.emit('users loaded', { users: users })

});

// загрузить сообщения

socket.on('load messages', function () {

socket.emit('messages loaded', { messages: messages })

});

// добавить нового пользователя в чат

socket.emit('new user', { name: users[users.length - 1] });

});

server.listen(port, function () {

console.log('app running on port ' + port);

})