Вы уже установили ноду в предыдущих уроках курса. Но если начали работать на другом компьютере или ноду съела собака, можно снова скачать её тут:

[Библиотека пакетов NPM](https://praktikum.yandex.ru/learn/web/courses/370a2c73-45bf-439f-a747-ef4e3c0db48f/sprints/1703/topics/af1c30a0-05e5-46b5-8570-6742270bc6d8/lessons/548a7bd4-8c6e-4846-8bdb-7ffe699ab718/)

# Создать сервер, настроить порт, установить запрос\ответ

const http = require('http');

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

console.log('Пришёл запрос!');

console.log(request);

console.log(response);

});

server.listen(3000);

# Объект запроса

const http = require('http');

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

console.log(req.url); *// /hello*

console.log(req.method); *// GET*

console.log(req.headers); *// здесь будут заголовки запроса*

console.log(req.body); *// а здесь тело запроса, но у GET запроса его нет*

});

# Объект ответа

Код статуса ответа и заголовки можно передать одним методом — writeHead:

Скопировать кодJAVASCRIPT

const http = require('http');

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

res.writeHead(200, {

'Content-Type': 'text/html; charset=utf8'

});

*// в методе end тоже можно передать данные*

res.end('<h1>Привет, мир!</h1>', 'utf8');

});

server.listen(3000);

# Переменные окружения

В Node.js это позволяют сделать переменные окружения. Они доступны из любой части программы, поэтому мы можем передать их при запуске сервера из терминала. Для этого перед командой запуска прописывают имена переменных и их значения:

Скопировать кодBASH

NODE\_ENV=production node index.js

*# NODE\_ENV - имя переменной окружения, а production — её значение*

Внутри скриптов переменные окружения хранятся в объекте process.env:

Скопировать кодJAVASCRIPT

if (process.env.NODE\_ENV !== 'production') {

console.log('Код запущен в режиме разработки');

}

# Обработка запроса

Когда приходит последний пакет, происходит событие end. Его и нужно обработать, чтобы превратить запрос в JSON-объект.

const http = require('http');

const { PORT = 3000 } = process.env;

const { BASE\_PATH } = process.env;

const mainPageMarkup = `

<!DOCTYPE html>

<html>

…………

</html>

`;

const submitSuccessMarkup = `

<!DOCTYPE html>

<html>

………………..

</html>

`;

const todos = [];

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

if (req.url === '/submit' && req.method === 'POST') {

let body = '';

req.on('data', (chunk) => {

body += chunk;

});

req.on('end', () => {

console.log(body); // например, здесь

// выделим всё, что идёт после знака равно, и добавим это в массив todos

todos.push(body.split('=')[1]);

console.log(todos);

res.writeHead(200, 'OK', {

'Content-Type': 'text/html'

});

res.end(submitSuccessMarkup);

});

}

if (req.url === '/' && req.method === 'GET') {

res.writeHead(200, {

'Content-Type': 'text/html'

});

res.end(mainPageMarkup);

}

});

server.listen(3000);

# Импорт модулей

Также мы можем импортировать модуль из других папок, задав относительный путь:

Скопировать кодJAVASCRIPT

const utils = require('../utils'); *// две точки означают переход в директорию на уровень выше*

const helpers = require('../../helpers');

# Экспорт модулей

*// utils.js*

const someFunction = () => {

console.log('Меня экспортировали');

};

const someValue = 42;

module.exports = {

someFunction,

someValue

};

При импорте мы получим объект module.exports. Останется только достать из него свойства - проще сделать это деструктуризацией:

*// index.js*

const { someFunction, someValue } = require('./utils');

# ES6 модули работают в ноде

Кроме того, чтобы ES6 модули заработали:

* ноду нужно запустить со специальным флагом -experimental-modules;
* все файлы должны иметь расширение .mjs.

Скопировать кодBASH

*# вот так ES6 модули работают*

node -experimental-modules index.mjs

Экспорт ES6 модулей выполняется иначе. Необходимо экспортировать каждое нужное значение директивой export:

Скопировать кодJAVASCRIPT

*// test.mjs*

export const theme = 'ES6-module';

export const method = (name) => { return `Здравствуйте, ${name}!` };

ES6-модули по умолчанию работают в строгом режиме, поэтому вам не нужно прописывать 'use strict'

# Модуль fs – работа с файловой системой

Это делают функцией readFile (англ. «прочитать файл»). Она работает асинхронно и принимает два аргумента: имя файла, из которого нужно прочитать данные, и колбэк. В колбэке описывают, что нужно сделать с прочитанными данными.

На вход колбэк принимает два аргумента: ошибку и данные файла.

const fs = require('fs');

fs.readFile('data.json', { encoding: 'utf8' }, (err, data) => { *// передали вторым аргументом объект опций. Он содержит свойство encoding, куда записывают кодировку данных*

if (err) {

console.log(err);

return;

}

console.log('data: ', data); *// здесь мы не вызываем метод toString, поскольку данные уже пришли в виде строки*

});

### Читать все файлы директории метод fs.readdir

Для этого есть метод fs.readdir. Он читает все файлы внутри директории. Первый аргумент метода — путь к директории. Второй — колбэк, в котором описано, что делать с полученными данными.

У колбэка также два параметра. Первый — ошибка, второй — массив имён найденных файлов:

Скопировать кодJAVASCRIPT

const fs = require('fs');

fs.readdir('.', (err, files) => {

if (err) {

console.log(err);

return;

}

console.log('data: ', files);

});

### Создавать папки метод fs.mkdir

С этим поможет метод fs.mkdir. Он принимает два параметра: имя новой папки и колбэк с единственным аргументом — ошибкой. Первым параметром можно передать имя папки вместе с путём, где нужно создать папку.

Скопировать кодJAVASCRIPT

const fs = require('fs');

fs.mkdir('incomingData/data', (err) => {

if (err) console.log(err);

});

### Записывать данные в файл метод fs.writeFile

Для этого есть метод fs.writeFile. Принимает три параметра:

* файл, куда нужно записать данные;
* сами данные, записанные строкой;
* колбэк для обработки ошибки.

Скопировать кодJAVASCRIPT

const fs = require('fs');

fs.writeFile('data.json', JSON.stringify([1, 2, 3]), (err) => {

if (err) console.log(err);

});

### Удалять файлы метод fs.unlink

Это делает метод fs.unlink. У него два параметра: имя файла и колбэк для обработки ошибок.

Скопировать кодJAVASCRIPT

const fs = require('fs');

fs.unlink('data.json', (err) => {

if (err) {

console.log(err);

return;

}

console.log('Файл удалён!');

});

### Много чего ещё

Остальные методы модуля fs по большей части работают аналогично. Так что, если захотите сделать с файлом что-то, чего мы не объясняли — почитайте документацию: вы без труда разберётесь.

[Ссылка на документацию модуля fs.](https://nodejs.org/api/fs.html)

# Промисы при работе с файлами

В 10 версии Node.js в модуле fs появилась поддержка промисов. Если вы пользуетесь промисами, колбэки передавать не нужно.

const fsPromises = require('fs').promises;

fsPromises.readFile('data.json', { encoding: 'utf8' })

.then((data) => {

console.log(data);

})

.catch(err => {

console.log(err);

});

[Ссылка на документацию к API для поддержки промисов](https://nodejs.org/api/fs.html#fs_fs_promises_api)

# Где находится модуль

Внутри любого модуля есть переменные \_\_filename и \_\_dirname. Они хранят имя файла модуля и путь к папке, где лежит модуль, соответственно.

Скопировать кодJAVASCRIPT

*// app.js*

console.log(\_\_filename); *// /usr/local/project/app.js*

console.log(\_\_dirname); *// /usr/local/project*

# Модуль path Как преобразовать путь

Модуль path хранит множество полезных методов для работы с путём к файлу. Задачу по преобразованию выполняет метод join. Он учитывает контекст, а потому не возникнет проблемы с разделителями, о которой мы говорили:

Скопировать кодJAVASCRIPT

*// read-file.js*

const fs = require('fs');

const path = require('path');

module.exports.readFile = () => {

const filepath = path.join(\_\_dirname, 'file.txt'); *// собрали абсолютный путь к файлу*

const file = fs.readFile('file.txt', { encoding: 'utf8' }, (err, data) => {

console.log(data);

});

};

Вот ещё несколько полезных методов модуля path:

Скопировать кодJAVASCRIPT

const fs = require('fs');

const path = require('path');

*// path.normalize убирает из пути лишние*

*// разделители и разрешает символы .. и .*

path.normalize('/foo/bar//baz/asdf/quux/..'); *// /foo/bar/baz/asdf*

*// path.dirname извлекает путь к папке из переданного пути*

path.dirname(process.mainModule.filename); *// /usr/local/my-project*

*// path.extname извлекает расширение файла*

path.extname('app.js'); *// .js*

[О других методах можете почитать в документации Node.js](https://nodejs.org/api/path.html)

Резюмируем: для работы с файловой системой нужны два модуля: fs и path. Первый содержит методы для работы с файлами, второй — для преобразования путей.

# Как работать с потоками?

Чтобы скомбинировать потоки чтения и записи, то есть превратить их в дуплексный поток, нужно при каждом событии data потока чтения вносить порцию данных в поток записи:

Когда данные полностью прочитаны, поток для чтения сгенерирует событие end. Его нужно обработать и сообщить потоку записи, что все данные пришли и ничего больше записывать не нужно. Для этого у потока записи есть метод end:

Последнее, что нужно сделать — обработать ошибку. В случае ошибки поток для чтения сгенерирует событие error. Его нужно обработать:

Скопировать кодJAVASCRIPT

const fs = require('fs');

const reader = fs.createReadStream('./in.txt', { encoding: 'utf8' });

const writer = fs.createWriteStream('./out.txt', { encoding: 'utf8' });

reader.on('data', (data) => {

writer.write(data);

});

reader.on('end', (data) => {

writer.end();

});

*// повесим обработчик события error*

reader.on('error', (err) => {

console.log(err);

});

## Метод pipe для работы с потоками

Чтобы организовать простейшую работу с двумя потоками, мы добавили три обработчика событий. Но код можно сделать заметно короче, если воспользоваться специальным методом для комбинирования потоков — pipe. Вам даже не нужно закрывать потоки и обрабатывать ошибки — вся эта логика уже описана внутри метода pipe:

Скопировать кодJAVASCRIPT

const fs = require('fs');

const reader = fs.createReadStream('./in.txt', { encoding: 'utf8' });

const writer = fs.createWriteStream('./out.txt', { encoding: 'utf8' });

reader.pipe(writer);

Метод pipe есть только у потоков для чтения — поток для записи передают как аргумент.

Напоследок разберём простой пример. Обращение пользователя к серверу — тоже поток для чтения, так что мы можем обработать этот запрос методом pipe:

Скопировать кодJAVASCRIPT

const http = require('http');

const fs = require('fs');

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

req.pipe(fs.createWriteStream(`./out-${Math.random()}.txt`));

});

server.listen(3000);

Данные, которые пользователь отправил в запросе, мы записываем этим кодом в текстовый файл со случайным именем.

Пример

const http = require('http');

const fs = require('fs');

const path = require('path');

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

const filePath = path.join(\_\_dirname, 'war-and-peace.txt');

const fileReader = fs.createReadStream(filePath, { encoding: 'utf8' });

res.writeHead(200, {

'Content-Type': 'text/plain'

});

fileReader.pipe(res);

});

server.listen(3000);

pipe — метод объекта чтения. Поток записи он принимает на вход:

fileReader.pipe(res);

# Отладка в браузере

Чтобы отлаживать код Node.js в браузере, выполните два шага:

1. установите в Яндекс.Браузер или в Google Chrome расширение [Node.js Inspector Manager](https://chrome.google.com/webstore/detail/nodejs-v8-inspector-manag/gnhhdgbaldcilmgcpfddgdbkhjohddkj);
2. запустите приложение с флагом --inspect: