|Настраиваем окружение

1.создаем папку mkdir blog

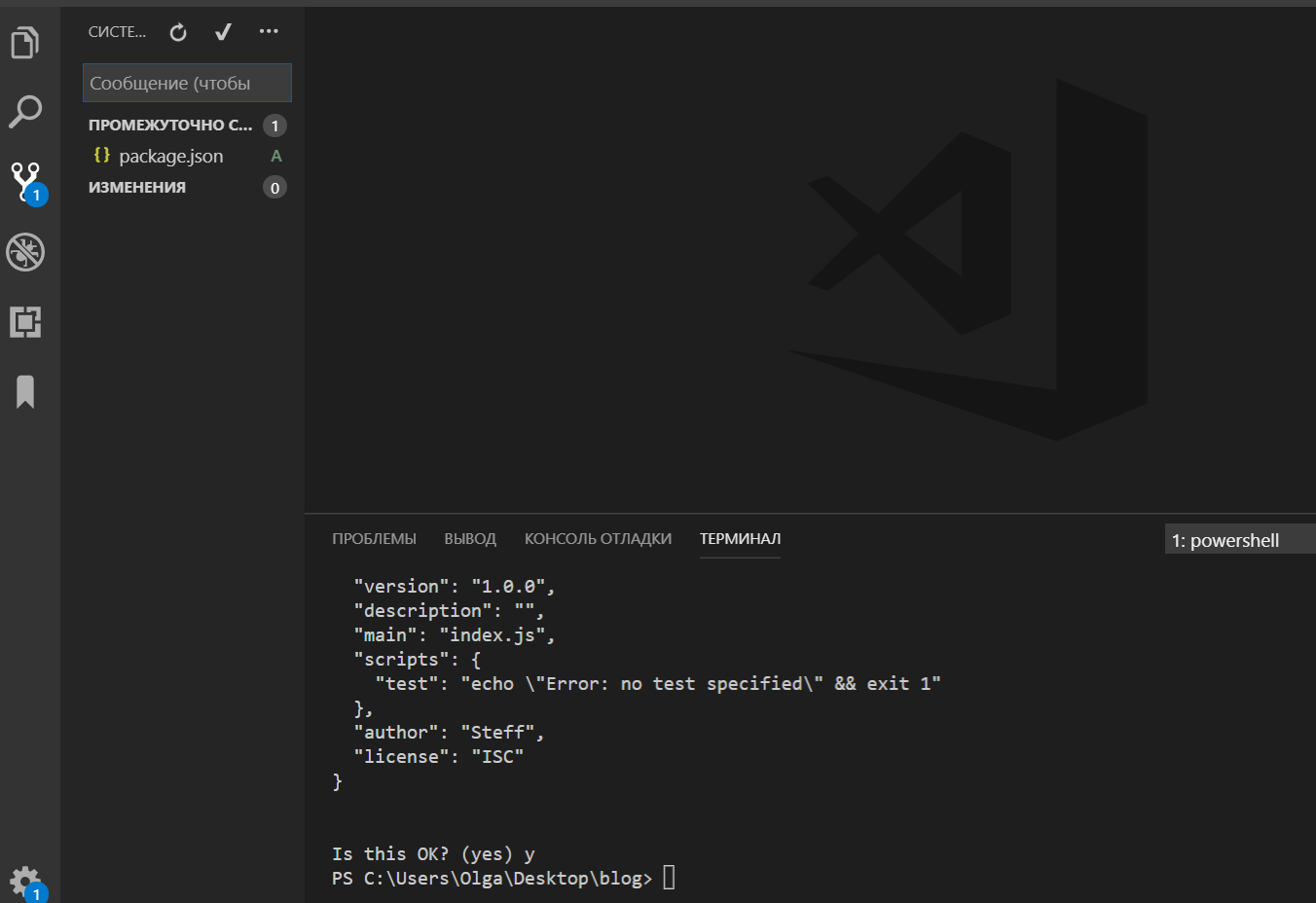
2.переходим в нее cd blog

3.инициализация git init

4.открываем проект в редакторе code .

5.Инициализируем пакетный менеджер

npm init



**Создать файл .gitignore в нем написать node\_modules**

# настройка eslint

# - стандартизатор и проверка кода

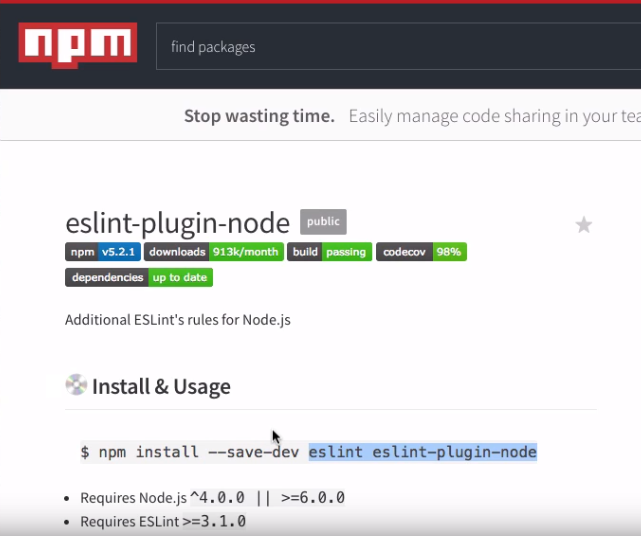
npm install --save-dev eslint eslint-plugin-node

***Должен быть установлен плагин Eslint устанавливается в редакторе***

***Или***

***Плагин импорта для ESLint***

npm i --save-dev eslint-plugin-import



* + создаем файл .eslintrc и вставим в него код с сайта npm .eslintrc.json(exempl)

{

"plugins": ["node"],

"extends": ["eslint:recommended", "plugin:node/recommended"],

"rules": {

"node/exports-style": ["error", "module.exports"],

"node/prefer-global/buffer": ["error", "always"],

"node/prefer-global/console": ["error", "always"],

"node/prefer-global/process": ["error", "always"],

"node/prefer-global/url-search-params": ["error", "always"],

"node/prefer-global/url": ["error", "always"]

}

}

# уprettier

"editor.tabSize": 2,"editor.formatOnSave": true,"prettier.printWidth": 100,"prettier.singleQuote": true,

"prettier.eslintIntegration": true,"prettier.stylelintIntegration": true,"prettier.trailingComma": "es5"

…

добавляем в package.json

"engines": {

"node": ">=10.14.1"

}

Package.json

{

"name": "blog",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Steff",

"license": "ISC",

"devDependencies": {

"eslint": "^5.15.3",

"eslint-plugin-node": "^8.0.1"

},

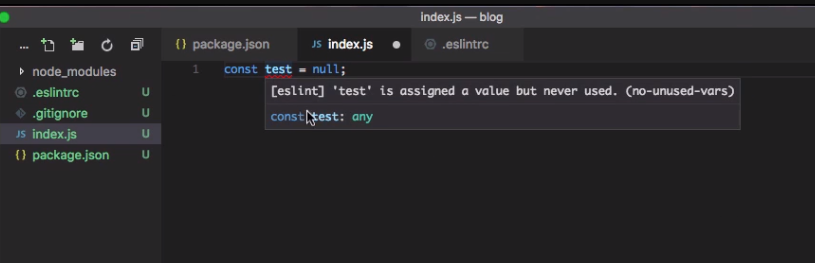
"engines": {

"node": ">=10.14.1"

}

}

Проверка



# Установка express

npm install express –save

Настройка express

<https://expressjs.com/ru/starter/hello-world.html>

var express = require('express');

var app = express();

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

res.send('Hello World!');

});

app.listen(3000, function () {

console.log('Example app listening on port 3000!');

});

Тот же код в ES6

*Код вписали в index.js.*

Приложение запускает сервер и слушает соединения на порте 3000. Приложение выдает ответ “Hello World!” на запросы, адресованные корневому URL (/) или **маршруту**. Для всех остальных путей ответом будет **404 Not Found**.

req (запрос) и res (ответ) являются теми же объектами, которые предоставляет Node, поэтому можно вызвать req.pipe(), req.on('data', callback) и выполнить любые другие действия, не требующие участия Express.

const express = require('express') // *подключение express*

const app = express(); //*создаем объект приложения*

const port = 3000

// *определяем обработчик для маршрута "/" и отправляем ответ*

app.get('/', (req, res) => res.send('Hello World!'))

*// начинаем прослушивать подключения на 3000 порту*

app.listen(port, () => console.log(`Example app listening on port ${port}!`))

eslintrc подчеркивает консоль, чтоб это исправить

в .eslintrc в rules

"no-console": 0

}

# nodemon

Запуск приложения lockalhost:3000

Чтобы перезапускалось автоматом устанавливаем инструмент **nodemon**

npm install --save-dev nodemon

Описываем секцию scripts в package.json. Для nodemon создадим секцию dev

"scripts": {

"start": "node index.js",

"dev": "nodemon index.js",

"test": "echo \"Error: no test specified\" && exit 1"

},

Теперь запуск **npm run dev**

# Работа с POST/GET запросами в Express — видео 2

# Шаблонизатор EJS

npm install ejs

1. Укажем express использовать шаблонизатор

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

1. По умолчанию все шаблоны будут искаться в папке views

Создадим:

* Папка views => файл index.ejs

1. Укажим рендерить шаблон index.ejs

Вместо

app.get('/', (req, res) => res.send('Hello World'))

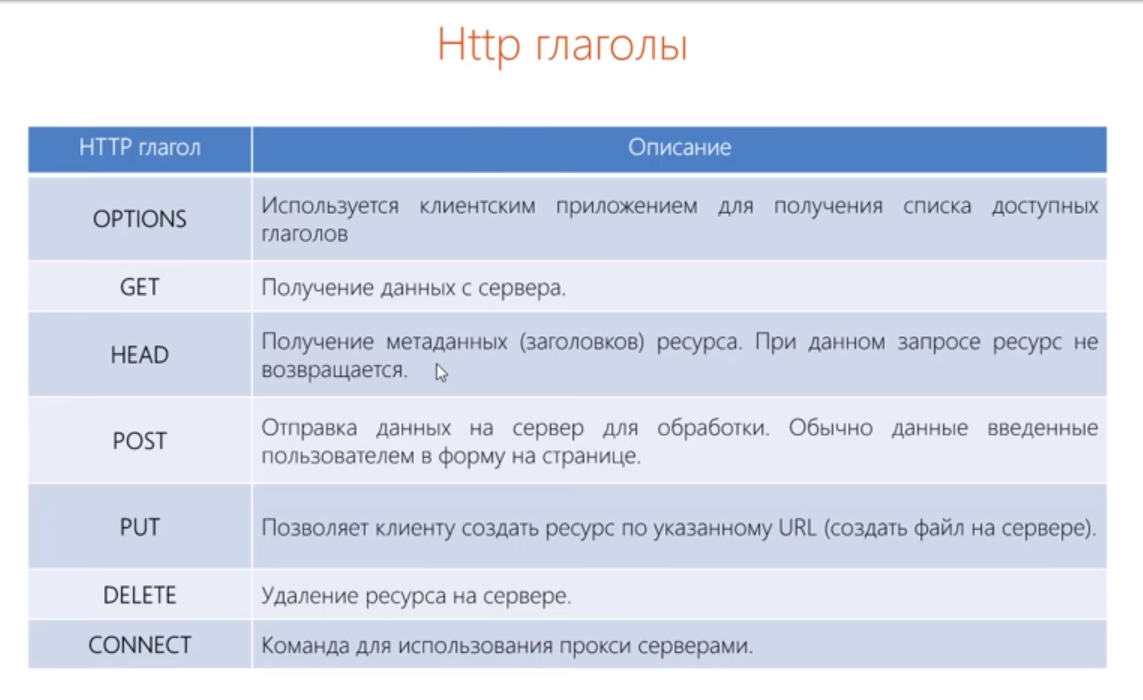
Пишем

app.get('/', (req, res) => res.render('index'))

1. Вывод данных. Данные передаются в рендере, вторым аргументом

* app.get('/', (req, res) => res.render('index', {data: data}));
* const data = "hi";
* <%= data %>
* Index.js
* Index.ejs

# POST/GET запросы

get –для адресной строки. Должно отправлять данные, но никак не изменять

post- для больших объемов и чтоб данные не отражались. Служит для высылания данных.

В REST API используют для создания новых засобов коллекции

1. Создаем create.ejs

* Create.ejs

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<form method="POST">

<input type="text" name="text">

<button type="submit">submit</button>

</form>

</body>

</html>

1. Ссылка для обработки в index.ejs

* <a href="/create">add</a>

1. Пишем оброботчик для ссылки в index.js

* app.get('/create', (req, res) => res.render('create'));

'/create'//то что в браузере

render('create'))// обработчик в папке views(шаблон)

Пишем оброботчик для POST запроса

Post-обьект

1. Установка **body-parser**

npm install body-parser

1. Подключаем его

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

1. Укажем его использование приложением

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

1. Обработчик

* app.post('/create', (req, res) =>{
* console.log(req.body);
* });

1. Вместо data создадим массив

* const express = require('express')
* const app = express()
* const port = 3000;
* var bodyParser = require('body-parser');
* app.set('view engine', 'ejs');
* app.use(bodyParser.urlencoded({ extended: true }));
* const arr = ['music', 'sun', 'bird'];
* app.get('/', (req, res) => res.render('index', {arr: arr}));
* app.get('/create', (req, res) => res.render('create'));
* app.post('/create', (req, res) =>{
* console.log(req.body);
* });
* app.listen(port, () => console.log(`Example app listening on port ${port}!`))

1. Делаем литератор для массива в шаблоне

* <ul>
* <% arr.forEach(function(item){ %>
* <li><%=item%></li>
* <% }); %>
* </ul>

## **Добавление элемента в массив через форму**

app.post('/create', (req, res) =>{

arr.push(req.body.text);

res.redirect('/')

console.log(req.body);

});

# | Подключаем MongoDB

* 1. Создадим конфигурационный файл в главном каталоге

Config.js

# Переменные окружения

**Переменные окружения или переменные среды (environment variables)**— это некие глобальные значения, расположенные на уровне операционной системы, доступные программам, например настройки системы.

Самой известной переменной, можно считать **PATH**. Операционная система использует значение этой переменной для того, чтобы найти нужные исполняемые файлы в командной строке или окне терминала. Например, когда мы выполняем в терминале команду **node**, **npm** или другую, именно переменная **PATH**подсказывает где искать исполняемые файлы. Об этом, подробнее рассказывается в этом [посте](https://coolestguidesontheplanet.com/add-shell-path-osx/" \t "_blank), на платформах linux и macos. Наверняка вы сталкивались с этой коварной переменной в начале своей карьеры Замечу, что значения в переменных окружения хранятся в виде строк. Поэтому старайтесь не использовать цифровые и булевы типы данных, иначе при записи они преобразуются к строкам. Помните об этом.

#### Проблема

В процессе разработки приложения, будь то клиентское или серверное, может понадобится использовать приватные данные, например секретный токен для запросов на сторонний сервер. Вот так:



[https://gist.github.com/Hydrock/74151288d3c0863db975827a14ac5c4d](https://gist.github.com/Hydrock/74151288d3c0863db975827a14ac5c4d" \t "_blank)

Но мы не можем сохранить такой код в git, иначе мы просто расскажем всему миру наш секретный ключ.

#### Время переменных окружения

Мы можем хранить секретные данные, настройки сборки и другие данные в переменных окружения. Программа на **nodejs**имеет к ним доступ.

В **nodejs** есть глобальный объект **process** (доступный из любого места программы, как window в браузере), хранящий информацию о текущем процессе. У этого объекта есть свойство **env**— оно и дает доступ к переменным окружения. Попробуйте запустить **node** в терминале и выполнить **console.log(process.env)**:



[https://gist.github.com/Hydrock/91d18e8c57a9572d07ac12c39de1056c](https://gist.github.com/Hydrock/91d18e8c57a9572d07ac12c39de1056c" \t "_blank)

Мы увидим как в терминал выведется объект со всеми значениями переменных окружения.

Теперь если представить, что наш секретный ключ уже находится в переменных окружения, то предыдущий пример можно переписать так:



[https://gist.github.com/Hydrock/b5ab3530bb20a2d85d2483bc84603dc3](https://gist.github.com/Hydrock/b5ab3530bb20a2d85d2483bc84603dc3" \t "_blank)

Мы получаем наш секретный ключ из окружения. Отличный пример 😂.

Еще, вы наверняка использовали или встречали конструкцию **node.env.NODE\_ENV === ‘production’** для определения режима сборки своего приложения.

Но как же установить эти переменные окружения?

#### Установка переменных окружения

Главной сложностью, является различие в способах установки переменных окружения в разных операционных системах и разных терминальных оболочках. В основном мы имеем две платформы: Windows и Linux (MacOS). Давайте посмотрим как это сделать в обоих.

[Классно про переменные окружения](https://medium.com/@hydrock/%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5-%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B2-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8-node-js-e9ca2131e6b6)

## Установим модуль для переменных окружения

npm install --save-dev cross-env

{

"name": "blog",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"start": "cross-env PORT=3001 node index.js",

"dev": "nodemon index.js",

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Steff",

"license": "ISC",

"devDependencies": {

"cross-env": "^5.2.0",

"eslint": "^5.15.3",

"eslint-config-prettier": "^4.1.0",

"eslint-plugin-node": "^8.0.1",

"nodemon": "^1.18.10"

},

"engines": {

"node": ">=10.14.1"

},

"dependencies": {

"body-parser": "^1.18.3",

"ejs": "^2.6.1",

"express": "^4.16.4"

}

}

const express = require('express')

const app = express()

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

const config = require('./config')

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

app.use(bodyParser.urlencoded({

extended: true

}));

const arr = ['music', 'sun', 'bird'];

app.get('/', (req, res) => res.render('index', {

arr: arr

}));

app.get('/create', (req, res) => res.render('create'));

app.post('/create', (req, res) => {

arr.push(req.body.text);

res.redirect('/')

console.log(req.body);

});

app.listen(config.PORT, () => console.log(`Example app listening on port ${config.PORT}!`))

* config.js
* module.exports = {
* PORT: process.env.PORT || 3000
* };

# MongoDB

[Установка и начало работы с MongoDB на Windows](https://metanit.com/nosql/mongodb/1.2.php)

[Хорошая установка(видео](https://youtu.be/TIH3dcZjrzk))

[Что такое mongoose](https://code.tutsplus.com/ru/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527)

[Mongoose для mongoDB](http://stepansuvorov.com/blog/2012/11/mongoose-%D0%B4%D0%BB%D1%8F-mongodb/)

Создаем файл database.js взято с файла index.js

const config = require('./config')

Устанавливаем **mongoose odm**

**npm install mongoose**

* database.js

подключаем

* const mongoose = require('mongoose');

**database.js**

module.exports = () => {

return new Promise((resolve, reject) => {

mongoose.Promise = global.Promise;

mongoose.set('debug', true);

mongoose.connection

.on('error', error => reject(error))

.on('close', () => console.log('Database connection closed.'))

.once('open', () => resolve(mongoose.connections[0]));

mongoose.connect(config.MONGO\_URL, {

useMongoClient: true

});

});

};

}

**[Обьект обещания](https://php-academy.kiev.ua/blog/javascript-callbacks-promises-and-async-functions-part-)**

**config.js**

* module.exports = {
* PORT: process.env.PORT || 3000,
* MONGO\_URL: 'mongodb://localhost/blog'
* };

**Создаем файл app.js. Перенести часть с index.js**

**app.js**

const express = require('express');

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

const app = express();

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

app.use(bodyParser.urlencoded({

extended: true

}));

const arr = ['hello', 'world', 'test'];

app.get('/', (req, res) => res.render('index', {

arr: arr

}));

app.get('/create', (req, res) => res.render('create'));

app.post('/create', (req, res) => {

arr.push(req.body.text);

res.redirect('/');

});

***Дописать***

module.exports = app;

**index.js**

* app.listen(config.PORT, () =>
* console.log(`Example app listening on port ${config.PORT}!`)
* );

**+**

* const app = require('./app');
* const database = require('./database');
* const config = require('./config');

**нихера не ясно**

* const app = require('./app');
* const database = require('./database');
* const config = require('./config');
* database()
* .then(info => {

console.log(`Connected to ${info.host}:${info.port}/${info.name}`);

app.listen(config.PORT, () =>

console.log(`Example app listening on port ${config.PORT}!`)

);

* })
* .catch(() => {

console.error('Unable to connect to database');

process.exit(1);

* });

# |Создаём первую модель

Модель

Создаем модель поста

Папка models -> post.js

Подключаем mongoose

const mongoose = require('mongoose');

Пишем схему

const Schema = mongoose.Schema;

const schema = new Schema({

title: {

type: String,

required: true

},

body: {

type: String

}

});

Экспорт

module.exports = mongoose.model('Post', schema);

В app.js подключаем модель поста

const Post = require('./models/post');

Вместо push

arr.push(req.body.text);

Post

Post.create({

title: title,

body: body

})

*Этот объект промис , поэтому его можно записать как, добавив метод*

Post.create({

title: title,

body: body

}).then(post => console.log(post.\_id))

[Реструктаризация](http://jsraccoon.ru/es6-destructuring)-поля объекта становятся переменными

const {

title,

body

} = req.body;

В creat.js

<body>

<form method="POST">

<input type="text" name="title">

<br>

<textarea name="body" cols="30" rows="7" name="body"></textarea>

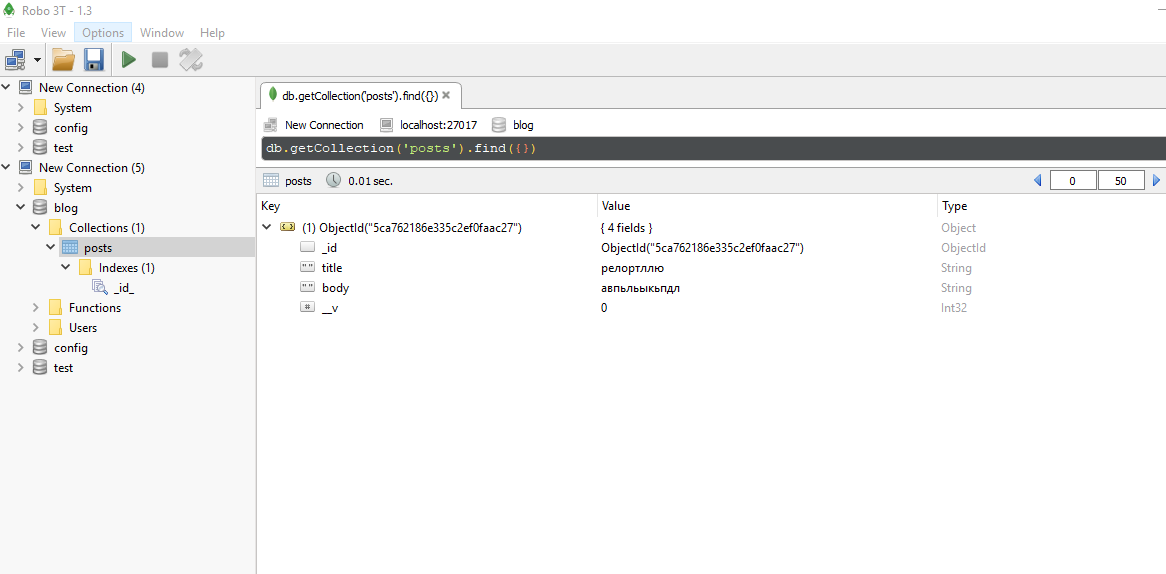
<br>

<button type="submit">submit</button>

</form>

</body>

**Результат**

****

Вернемся к модели и добавим timestamps

Добавляет дату создания и обновления

сonst schema = new Schema({

title: {

type: String,

required: true

},

body: {

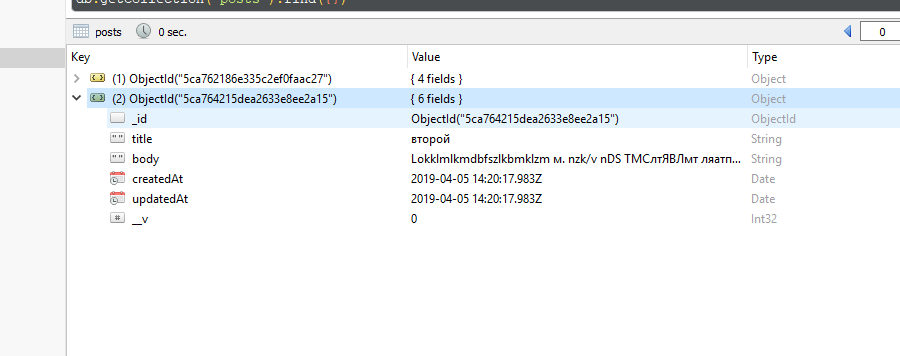
type: String

}

}, {

timestamps: true

});



***Чтоб не было кривых ID***

***А был JSON***

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const schema = new Schema({

title: {

type: String,

required: true

},

body: {

type: String

}

}, {

timestamps: true

});

schema.set('toJSON', {

virtuals: true

});

module.exports = mongoose.model('Post', schema);

Post.create({

title: title,

body: body

}).then(post => console.log(post.\_id))

Post.create({

title: title,

body: body

}).then(post => console.log(post.id))

**Выведем коллекцию в наш индекс**

**app.js**

app.get('/', (req, res) => {

Post.find({}).then(posts => {

res.render('index', {

posts: posts

});

})

});

**index.ejs**

<ul>

<% posts.forEach(function(post){ %>

<li>

<%= post.title%>

<br>

<%= post.body%>

</li>

<% }); %>

</ul>

# | Создание Layout и хостинг статики

**Создание loyout**

views - > папка loyout -> : footer.ejs &header.ejs

переносим нужное с index.ejs. Распределяем по двум новым

подключаем их в index.ejs

<% include layout/header.ejs%>

<% include layout/footer.ejs%>

Также обработать файл create.ejs

Создаем папку public. **В ней будут хранится все наши стили и скрипты**

**В ней папки и в них:** images, javascripts(scripts.js), stylesheets(styles.css)

**В node.js express нужно указать чтоб он хостил папку**

Для этого подключим модель PATH в app.js

const path = require('path');

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

Подключаем скрипт в нашем футере

<script src="/javascripts/scripts.js"></script>

Подключаем стили в хедере

<link rel="stylesheet" href="stylesheets/styles.css">

Ставим jquery через npm

**npm install jquery**

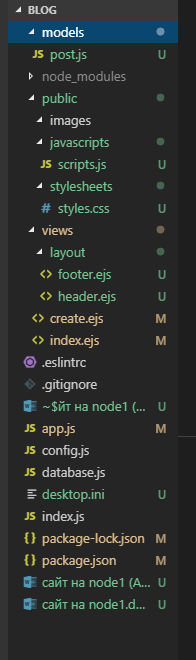
Укажем express что он должен отдавать файл jquery

app.use('/javascripts', express.static(path.join(\_\_dirname, 'node\_modules', 'jquery', 'dist')));

и footer

<script src="/javascripts/jquery.min.js"></script>

Удаляем лишние

До удаления

**app.js**

const express = require('express');

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

const path = require('path');

const Post = require('./models/post');

const app = express();

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

app.use(bodyParser.urlencoded({

extended: true

}));

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

app.use('/javascripts', express.static(path.join(\_\_dirname, 'node\_modules', 'jquery', 'dist')));

// const arr = ['hello', 'world', 'test'];

// app.get('/', (req, res) => res.render('index', {

// arr: arr

// }));

app.get('/', (req, res) => {

Post.find({}).then(posts => {

res.render('index', {

posts: posts

});

})

});

app.get('/create', (req, res) => res.render('create'));

app.post('/create', (req, res) => {

const {

title,

body

} = req.body;

Post.create({

title: title,

body: body

}).then(post => console.log(post.\_id))

res.redirect('/');

});

module.exports = app;

**index.js**

<% include layout/header.ejs%>

<a href="/create">add</a>

<ul>

<% posts.forEach(function(post){ %>

<li>

<%= post.title%>

<br>

<%= post.body%>

</li>

<% }); %>

</ul>

<% include layout/footer.ejs%>

После чистки удалили файл create.js

app.js

const express = require('express');

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

const path = require('path');

// const Post = require('./models/post');

const app = express();

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

app.use(bodyParser.urlencoded({

extended: true

}));

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

app.use('/javascripts', express.static(path.join(\_\_dirname, 'node\_modules', 'jquery', 'dist')));

app.get('/', (req, res) => {

res.render('index');

});

module.exports = app;

// const arr = ['hello', 'world', 'test'];

// app.get('/', (req, res) => res.render('index', {

// arr: arr

// }));

// app.get('/', (req, res) => {

// // Post.find({}).then(posts => {

// // // res.render('index', {

// // // posts: posts

// // // });

// // })

// });

// app.get('/create', (req, res) => res.render('create'));

// app.post('/create', (req, res) => {

// const {

// title,

// body

// } = req.body;

// Post.create({

// title: title,

// body: body

// }).then(post => console.log(post.\_id))

// res.redirect('/');

// });

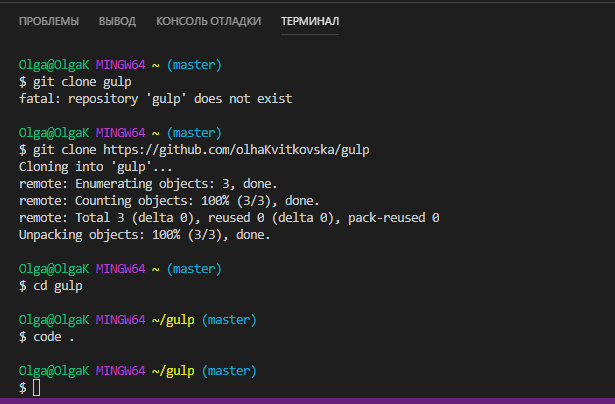
Все закомментированное можно удалить, но мне жалко

# | Настройка окружения для вёрстки страницы, Gulp

Делаем в новом проэкте а потом прикручиваем.

# git

[Начало работы с git](https://git-scm.com/book/ru/v1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-Git-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F)

****

C:\Users\Olga\Desktop\cmder\_mini (master -> origin)

λ **git clone https://github.com/olhaKvitkovska/gulp**

fatal: destination path 'gulp' already exists and is not an empty directory.

C:\Users\Olga\Desktop\cmder\_mini (master -> origin)

λ **cd gulp**

C:\Users\Olga\Desktop\cmder\_mini\gulp (master -> origin) (gulpfornode@1.0.0)

λ **code .**

C:\Users\Olga\Desktop\cmder\_mini\gulp (master -> origin) (gulpfornode@1.0.0)

λ

Создаем package.json

Подключаем eslint

В package.json указать версию node

"engines": {

"node": ">=10.14.1"

},

Создаем gulpfile.js

Устанавлевае gulp

npm install --save-dev gulp

Подключаем gulp

const gulp = require('gulp');

Дефолтная задача которая будет запускать таски

gulp.task('default', () => {

});

**SASS**

Создаем две папки и в них: dev(папка scss и файл main.scss), dist

Установим плагин gulp-sass

npm install gulp-sass

npm install gulp-autoprefixer

const sass = require('gulp-sass');

const autoprefixer = require('gulp-autoprefixer');

Cssnano для сжатия

npm install gulp-cssnano

const cssnano = require('gulp-cssnano');

Создаем задачу для sass

function css() {

return gulp

.src('./dev/scss/\*\*/\*.scss')

.pipe(sass())

.pipe(

autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], {

cascade: true

})

)

.pipe(cssnano())

.pipe(gulp.dest('dist/css'))

// .pipe(browserSync.stream());

}

Указываем задачу в default

gulp.watch('dev/scss/\*\*/\*.scss', css);

полный gulpfile.js

const gulp = require('gulp');

const sass = require('gulp-sass');

const autoprefixer = require('gulp-autoprefixer');

const cssnano = require('gulp-cssnano');

function css() {

return gulp

.src('./dev/scss/\*\*/\*.scss')

.pipe(sass())

.pipe(

autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], {

cascade: true

})

)

.pipe(cssnano())

.pipe(gulp.dest('dist/css'))

}

gulp.watch('dev/scss/\*\*/\*.scss', css);

exports.default = gulp.series(css);

Добавим сервер

Создаем index.html в папке dist

Подключаем наши стили

Устанавливаем browser – sync

npm i browser-sync

const browserSync = require('browser-sync').create();

function serve() {

browserSync.init({

server: {

baseDir: './dist'

},

notify: false

});

.pipe(browserSync.stream());

gulp.watch('dist/\*.html').on('change', browserSync.reload);

exports.default = gulp.series(serve, css);

**Полный gulpfile.js**

const gulp = require('gulp');

const sass = require('gulp-sass');

const autoprefixer = require('gulp-autoprefixer');

const cssnano = require('gulp-cssnano');

const browserSync = require('browser-sync').create();

function css() {

return gulp

.src('./dev/scss/\*\*/\*.scss')

.pipe(sass())

.pipe(

autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], {

cascade: true

})

)

.pipe(cssnano())

.pipe(gulp.dest('dist/css'))

.pipe(browserSync.stream());

}

function serve() {

browserSync.init({

server: {

baseDir: './dist'

},

notify: false

});

gulp.watch('dev/scss/\*\*/\*.scss', css);

gulp.watch('dist/\*.html').on('change', browserSync.reload);

}

exports.default = gulp.series(serve, css);

# | Верстаем базовый шаблон

Подключаем gulp-plumber

npm install --save-dev gulp-plumber

const plumber = require('gulp-plumber');

function css() {

return gulp

.src('./dev/scss/\*\*/\*.scss')

.pipe(plumber())

.pipe(sass())

.pipe(

autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], {

cascade: true

})

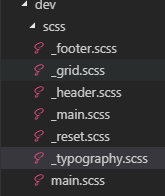
)

.pipe(cssnano())

.pipe(gulp.dest('dist/css'))

.pipe(browserSync.stream());

}



# | Перенос шаблона в основной проект, хеш для статики

# ДОКУМЕНТАЦИЯ

[Сокет](https://socket.io/docs/)

[Express](https://expressjs.com/)

[Шаблонизатор](https://ejs.co/)

<https://www.npmjs.com/>

<https://js-node.ru/>

<https://metanit.com/web/nodejs/3.3.php>

<https://www.bootstrapcdn.com/>

<https://metanit.com/nosql/mongodb/>

<https://mongoosejs.com/>