

Фреймворк Express

**Установка и обработка GET**

Установим модуль express:

npm i express

Пример создания маршрутов:

const express = require('express')

const process = require('process')

const PORT = process.env.PORT || 3000

let app = express()

app.get ('/', (request, response) => {

response.send ('<h1>GET</h1>')

})

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

response.send ('<h1>ABOUT</h1>')

})

app.listen(PORT)

Вызвав express, мы создаем объект приложения – app. Объект app будет обрабатывать все запросы. Для обработки Get запросов используется метод get. Ему передаем маршрут и функцию обработчик. Вызывая listen, мы запускаем сервер.

Метод send у response отправляет ответ на сайт.

**Отправка ответа**

Для отправки ответа в виде Buffer, строки, js объекта и сразу массива данных следует использовать send:

app.get ('/', (request, response) => {

response.send ({id: 12, name: 'Alex'})

})

Отправить сразу файл можно через метод sendFile, притом нужен лишь путь к файлу:

const path = require ('path')

app.get ('/', (request, response) => {

response.sendFile (path.join (\_\_dirname, 'server', 'public', 'index.html'))

})

Для отправки статус-кода есть метод sendStatus:

app.get ('/', (request, response) => {

response.sendStatus (404)

})

На странице будет стандартное сообщение об ошибке. Если нужно указать свое сообщение, то следует вызвать status(), а после у полученного объекта вызвать send с сообщением.

app.get ('/', (request, response) => {

response.status (404).send (`

<img src="https://cs.pikabu.ru/images/big\_size\_comm/2012-09\_3/13477242899764.jpg">

`)

})

**Переадресация**

Метод redirect позволяет переадресовывать. Можно переадресовывать по url:

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

res.redirect ('https://www.chitai-gorod.ru/')

})

Можно переадресовывать и относительного корня:

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

res.redirect ('/')

})

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

res.send ('<h1>ROOT</h1>')

})

По умолчанию статус будет 302, т.е. ресурс доступен временно на новом адресе. Можно первым указать статус код:

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

res.redirect (301, '/')

})

**Маршрутизация**

Для обработки запросов к конкретным маршрутам используются следующие методы:

* use
* get
* post
* put
* delete

Первым параметром они принимают шаблон адрес, а вторым функцию обработчик, который сработает, если путь совпадет. Порядок объявления маршрута важен, поэтому следует объявлять от частных шаблонов, к примеру: /about и заканчивать общим, например /, т.к. Express может посчитать, что /about есть /.

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

res.send ('<h1>About</h1>')

})

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

res.send ('<h1>Shop</h1>')

})

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

res.send ('<h1>Root</h1>')

})

В шаблонах можно использовать регулярные выражения!

app.get ('/bo+k(.html)?', (req, res) => {

res.send ('<h1>Book</h1>')

})

**Middleware**

Получая запрос, Express прогоняет его по *конвейеру обработки*. В этом конвейере лежат все вызовы методов типа get, которые называются *middleware*.

app.get ('/', (request, response) => {

response.send ('<h1>MAIN</h1>')

})

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

response.send ('<h1>ABOUT</h1>')

})

app.get ('/shop', (request, response) => {

response.send ('<h1>SHOP</h1>')

})

Здесь конвейер обработки состоит из вызовов get, которые будут вызываться, если совпал url. Конвейером можно управлять через метод use:

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

console.log ("Middleware 1")

next ()

})

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

console.log ("Middleware 2")

let rnd = Math.random()

console.log (rnd)

if (rnd > 0.5) {

next ()

} else {

response.send ('Lucky')

}

})

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

console.log ("Route /")

response.send ("Hello")

})

По сути метод use – это обработчик, который может вызвать следующий обработчик через метод next() и он будет вызван во всех запросах. Он удобен, когда нужно сделать что-либо перед отправкой ответа, например логирование:

const express = require("express")

const path = require('path')

const fs = require('fs')

const app = express()

app.use ((req, res, next) => {

let file = path.join(\_\_dirname, 'server', 'server.log')

let data = `${req.url}; ${req.method}; ${new Date()}\n`

fs.appendFile (file, data, () => {})

next ()

})

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

res.send ('<h1>ROOT</h1>')

})

app.listen(3000)

**Статические файлы**

То, что мы написали в прошлой главе можно заменить одним вызовом метода – express.static:

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

Компонент static, работает со всеми типами данных. В идеале нам нужно еще обрабатывать ошибки. В таком случаи поступить можно так:

const pages = [

'/',

'/about'

]

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

app.use ((request, response, next) => {

pages.forEach (el => {

if (el !== request.url) {

response.sendFile (path.join (\_\_dirname, 'public', 'error.html'))

}

})

})

**Передача данных через параметры строки запроса**

Очень часто встречается передача данных через *параметры* *строки запроса*, это когда в строке после url идут параметры и их значения.

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

console.log (req.query)

res.send (`<h1>${req.query.id}, ${req.query.name}</h1>`)

})

Обратимся по <http://localhost:3000/?id=123&name=Alex>

Передать можно объект:

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

console.log (req.query)

res.send (`

<h1>${req.query.user.id}, ${req.query.user.name}</h1>

<h1>${req.query.date}</h1>

`)

})

Обратимся по [http://localhost:3000/?user[id]=123&user[name]=Alex&date=18.11.19](http://localhost:3000/?user%5bid%5d=123&user%5bname%5d=Alex&date=18.11.19)

**Параметры маршрута**

*Параметры маршрута* похожи на параметры строки запроса, только первые это часть url и служит для маршрутизации, а вторые более условны и служат для передачи каких-либо данных.

Когда делаешь магазин информацию о пользователе, следует передавать через параметры строки запроса, а указывать категории товаров следует через параметры маршрута.

Параметры маршрута обозначаются символом :, но в браузере мы их не указываем.

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

res.send (`

<h1>Store</h1>

<h3>${req.params ['category']}</h3>

<p>

<strong>id</strong> - ${req.query['id']}

</p>

`)

})

**Класс Router**

Задавать все маршруты в одном месте неудобно, когда у нас средний или большой сайт. К примеру, есть сайт с магазином. По-хорошему все маршруты, относящиеся к магазину следует вынести в отдельный «маршрутизатор» и после соединить с основным.

Для этого есть класс Router:

const router = express.Router()

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

res.send ('<h1>All Product</h1>')

})

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

res.send ('create product')

})

router.get ('/:category', (req, res) => {

res.send (`Category ${req.params ['category']}`)

})

И соединяем этот маршрут с основным:

const app = express()

app.use ('/store', router)

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

res.send ('<h1>Main Title</h1>')

})

**Post запросы**

Для обработки post-запросов используют метод post. Вторым параметром ему нужно передать парсер ответа (есть стандартная реализация для json – express.json()), а сам ответ притом лежит в request.body.

const express = require('express')

const path = require('path')

const app = express()

const jsonParser = express.json()

app.post ("/user", jsonParser, (req, res) => {

console.log (req.body)

if (!req.body) return res.sendStatus(400)

res.json (req.body)

})

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

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

})

app.listen (3000)

Вот html страницы. В данном случае мы используем ajax.

<!DOCTYPE html>

<html>

<head>

<title>Регистрация </title>

<meta charset="utf-8" />

</head>

<body>

<h1>Регистрация </h1>

<form name="reg">

<label>Имя</label></br>

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

</br>

<label>Возраст</label></br>

<input type="number" name="age" /></br>

</br>

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

</form>

<script>

document

.getElementById ("submit")

.addEventListener ("click", async (e) => {

e.preventDefault ()

let form = document.forms.reg

let user = {

name: form.elements.name.value,

age: form.elements.age.value

}

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

method: 'POST',

headers: {

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

},

body: JSON.stringify(user)

})

let result = await response.json()

console.dir (result)

})

</script>

</body>

<html>

**Итого**

В этой главе были рассмотрены все основные возможно express, это:

* маршрутизатор
* обработка форм
* статические файлы
* параметры запроса и параметры строки запроса
* создали Rest API