

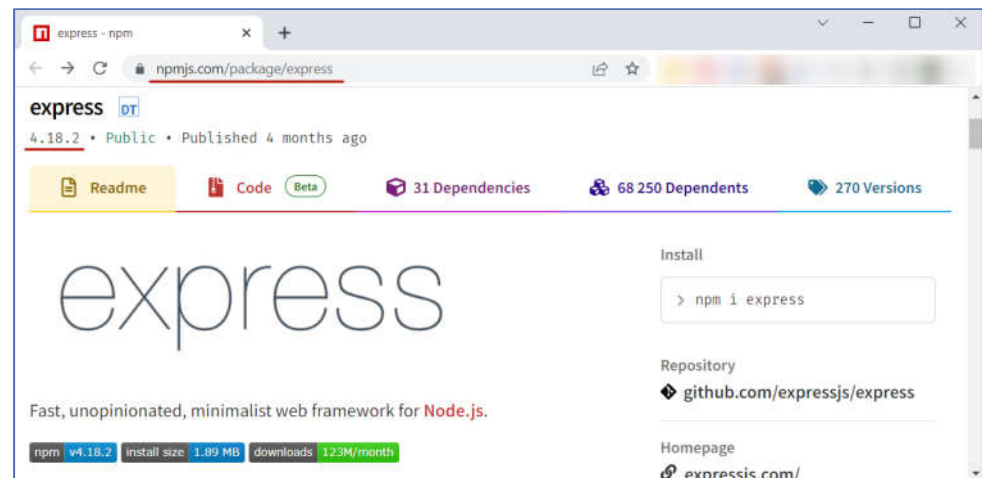
ПРОГРАММИРОВАНИЕ В INTERNET

EXPRESS

Express

=

это веб-фреймворк, который предоставляет ряд готовых абстракций и упрощает создание сервера (обработка форм, работа с cookie, CORS и т.д. Он использует модуль http.



[документация](#)

Создание сервера. Промежуточные обработчики (middleware)

```
const express = require("express"); // npm install express
const app = express();
```

```
app.use((req, res, next)=>{           //middleware
  console.log('handler 01');
  next();
});
```

```
app.use((req, res, next)=>{           //middleware
  console.log('handler 02');
  next();
});
```

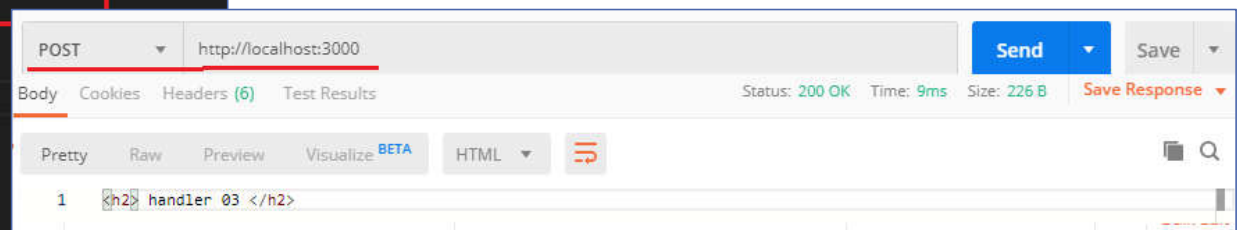
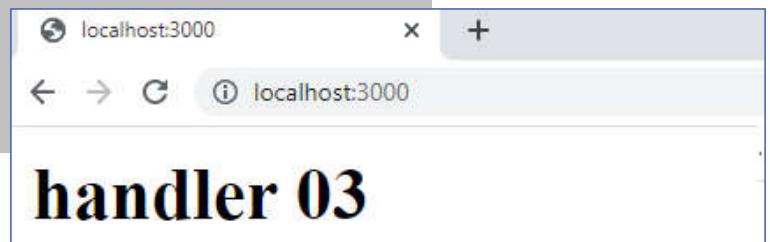
```
app.use((req, res, next)=>{           //middleware
  console.log('handler 03');
  res.send("<h2> handler 03 </h2>");
});
```

```
app.listen(3000);
```

Когда сервер получает запрос, этот запрос передается в **конвейер обработки запроса**. Можно встроить в конвейер обработки запроса на любом этапе функцию **middleware** (промежуточные обработчики). Для этого применяется метод **app.use()**.

```
D:\PSCA\Lec17_express>node 17-01
```

```
handler 01
handler 02
handler 03
```



Middleware

Функция, которая передается в `app.use()`, принимает три параметра:

- `request`: объект запроса;
- `response`: объект ответа;
- `next`: следующая в конвейере обработки запроса функция.

```
app.use('/path', (request, response, next) => {  
  // ...  
})
```

Функции `middleware` могут сопоставляться с определенными маршрутами.

Middleware

Может:

- выполнять любой код;
- изменять объекты запроса и ответа;
- заканчивать цикл обработки, отправив ответ (`res.send()`)
- вызывать следующий middleware из очереди.

Обязан:

- завершить запрос (`res.send()`);
- или вызвать следующий middleware (`next()`) .

Типы middleware

1. Промежуточные обработчики уровня приложения

```
app.use('/', (req, res, next) => {  
  console.log('application-level middleware');  
  next();  
});
```

2. Промежуточные обработчики уровня роутера

```
let router = express.Router();  
router.use('/', (req, res, next) => {  
  console.log('router-level middleware');  
  next();  
})
```

Типы middleware

3. Встроенные промежуточные обработчики

```
app.use(express.static('./public'));
```

4. Сторонние промежуточные обработчики

```
const cookieParser = require('cookie-parser');  
app.use(cookieParser());
```

5. Промежуточные обработчики для обработки ошибок

```
app.use((err, req, res, next) => {  
  console.error('error-handler', err);  
  res.status(500).send('error handler');  
});
```

Маршрутизация

```
app.METHOD(path, handler)
```

Приложение «прослушивает» запросы, соответствующие указанным **маршрутам** и **методам**, и когда оно обнаруживает совпадение, оно вызывает указанную **функцию обратного вызова**.

- ❗ При наличии нескольких функций обратного вызова важно указать `next` в качестве параметра функции обратного вызова, а затем вызвать `next()` в теле функции, чтобы
 - передать управление следующему обратному вызову.

Маршрутизация (методы)

Метод маршрута является производным от одного из методов HTTP и присоединяется к экземпляру express.

```
app.get('/', (req, res, next) => { });  
app.post('/', (req, res, next) => { });  
app.delete('/', (req, res, next) => { });
```

Существует специальный метод маршрутизации `app.all()`, используемый для отлова запросов всех методов HTTP-запросов.

```
app.all('/', (req, res, next) => { });
```

Для методов, которые преобразуются в недопустимые имена переменных JavaScript, используется нотация в квадратных скобках.

```
app['m-search']('/', (req, res, next) => { });
```

Маршрутизация (методы)

- checkout
- copy
- delete
- get
- head
- lock
- merge
- mkactivity
- mkcol
- move
- m-search
- notify
- options
- patch
- post
- purge
- put
- report
- search
- subscribe
- trace
- unlock
- unsubscribe

```
const express = require("express"); // npm install express
const app = express();

app.use((req, res, next)=>{console.log('handler 01'); next();}); // middleware
app.use((req, res, next)=>{console.log('handler 02'); next();}); // middleware
app.use((req, res, next)=>{console.log('handler 03'); next();}); // middleware
```

```
app.get('/', (req, res)=>{ // обработка get-запросов
  console.log('get / handler 04');
  res.send("<h2> handler 04 </h2>");
})
```

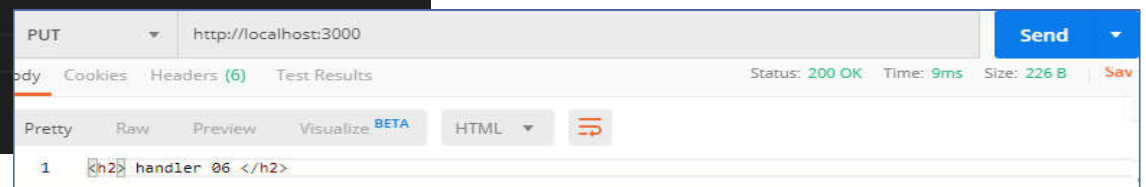
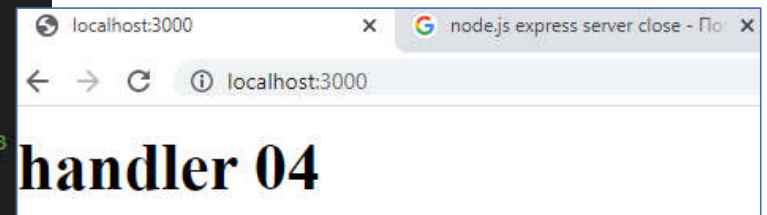
```
app.post('/', (req, res)=>{ // обработка post-запросов
  console.log('post / handler 05');
  res.send("<h2> handler 05 </h2>");
})
```

```
app.put('/', (req, res)=>{ // обработка put-запросов
  console.log('put / handler 06');
  res.send("<h2> handler 06 </h2>");
})
```

```
app.delete('/', (req, res)=>{ // обработка delete-запросов
  console.log('delete / handler 07');
  res.send("<h2> handler 07 </h2>");
})
```

```
var server = app.listen(3000); // = http.Server
```

```
D:\PSCA\Lec17_express>node 17-02
handler 01
handler 02
handler 03
get / handler 04
handler 01
handler 02
handler 03
put / handler 06
```



```

const express = require('express');
const app = express();

app.get('/', (req, res, next) => {
  console.log('get /');
  next();
});

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

app.put('/', (req, res, next) => {
  console.log('put /');
  next();
});

app.delete('/', (req, res, next) => {
  console.log('delete /');
  next();
});

app.all('/', (req, res, next) => {
  console.log('all /, ', req.method);
  res.send('ALL: ' + req.method);
});

app.listen(3000);

```

The image displays a sequence of five overlapping browser developer tool screenshots, each showing an HTTP request to `http://localhost:3000/` and its response. The requests are for the following methods: GET, POST, PUT, DELETE, and PROPFIND. Each response body is `ALL: [method]`.

- GET**: Response is `ALL: GET`.
- POST**: Response is `ALL: POST`.
- PUT**: Response is `ALL: PUT`.
- DELETE**: Response is `ALL: DELETE`.
- PROPFIND**: Response is `ALL: PROPFIND`.

Below the screenshots, a terminal window shows the command to run the application:

```

PS D:\NodeJS\samples\cwp_17> node 17-05
get /
all /, GET
post /
all /, POST
put /
all /, PUT
delete /
all /, DELETE
all /, PROPFIND

```

Маршрутизация (путь)

Путь, для которого вызывается функция; может быть представлен в виде:

- 1) строки;
- 2) паттерна (преобразуется в регулярное выражение);
- 3) регулярного выражения.

Маршрутизация (путь – строка)

```
app.get('/', (req, res, next) => {  
  res.send('get / ');  
});
```

GET http://localhost:3000/

Params Authorization Headers (8) Body

Body Cookies (1) Headers (7) Test Results

Pretty Raw Preview Visualize

1 get /

GET http://localhost:3000/a

Params Authorization Headers (8) Body

Body Cookies (1) Headers (8) Test Results

Pretty Raw Preview Visualize

Cannot GET /a

Маршрутизация (путь – строка)

```
app.get('/home', (req, res, next) => {  
  res.send('get /home');  
});
```

GET ⌵ http://localhost:3000/

Params Authorization Headers (8) Body

Body Cookies (1) Headers (8) Test Results

Pretty Raw Preview Visualize

Cannot GET /

GET ⌵ http://localhost:3000/home

Params Authorization Headers (8) Body

Body Cookies (1) Headers (7) Test Results

Pretty Raw Preview Visualize

1 get /home

GET ⌵ http://localhost:3000/home2

Params Authorization Headers (8) Body

Body Cookies (1) Headers (8) Test Results

Pretty Raw Preview Visualize

Cannot GET /home2

Маршрутизация (путь – строка)

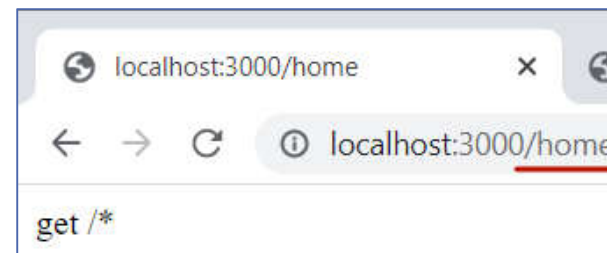
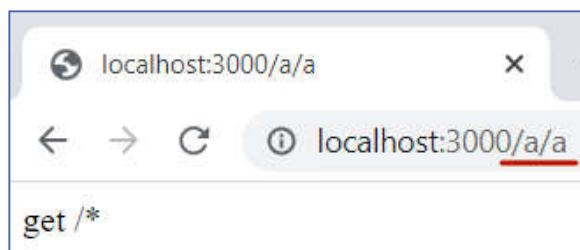
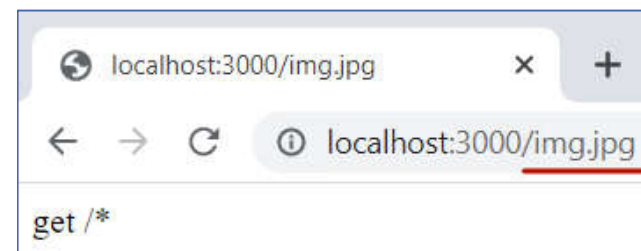
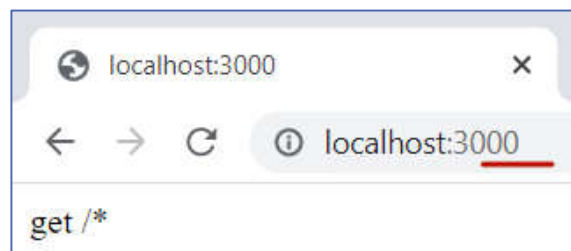
```
app.get('/image.png', (req, res, next) => {  
  res.send('get /image.png');  
});
```

<div>GET</div> <div>http://localhost:3000/</div> <div>ParamsAuthorizationHeaders (8)Body</div> <div>BodyCookies (1)Headers (8)Test Results</div> <div>PrettyRawPreviewVisualize</div> <div>Cannot GET /</div>	<div>GET</div> <div>http://localhost:3000/image.png</div> <div>ParamsAuthorizationHeaders (8)Body</div> <div>BodyCookies (1)Headers (7)Test Results</div> <div>PrettyRawPreviewVisualize</div> <div>get /image.png</div>	<div>GET</div> <div>http://localhost:3000/image.jpg</div> <div>ParamsAuthorizationHeaders (8)Body</div> <div>BodyCookies (1)Headers (8)Test Results</div> <div>PrettyRawPreviewVisualize</div> <div>Cannot GET /image.jpg</div>
---	--	---

Маршрутизация (путь – паттерн)

* – произвольный набор символов

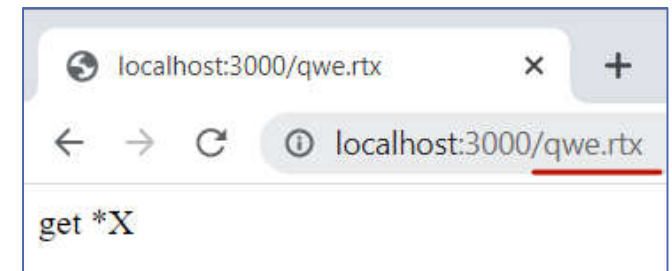
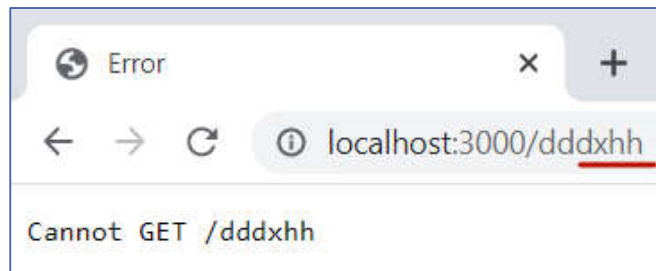
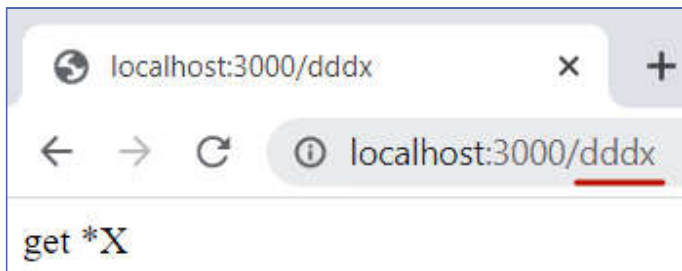
```
app.get('/*', (req, res, next) => {  
  res.send('get /*');  
});
```



Маршрутизация (путь – паттерн)

* – произвольный набор символов

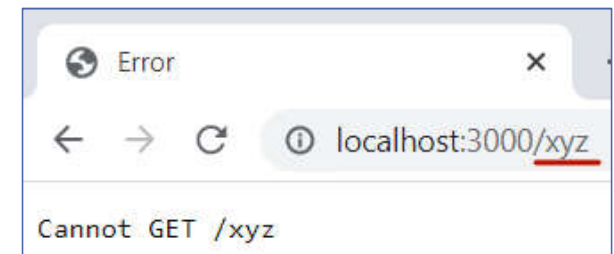
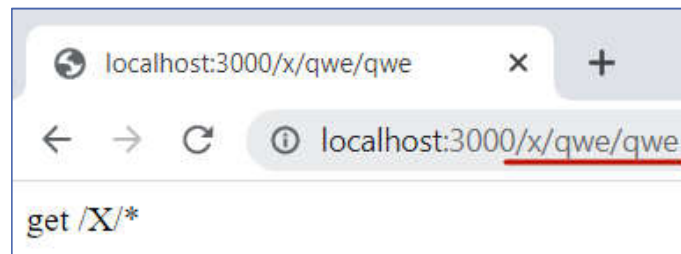
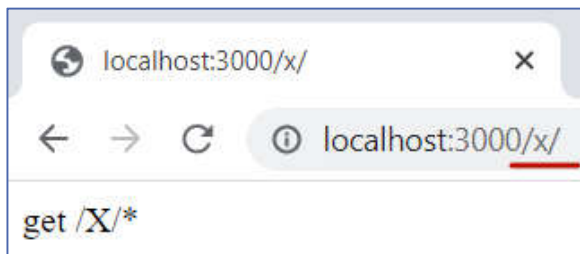
```
app.get('*X', (req, res, next) => {  
  res.send('get *X');  
});
```



Маршрутизация (путь – паттерн)

* – произвольный набор символов

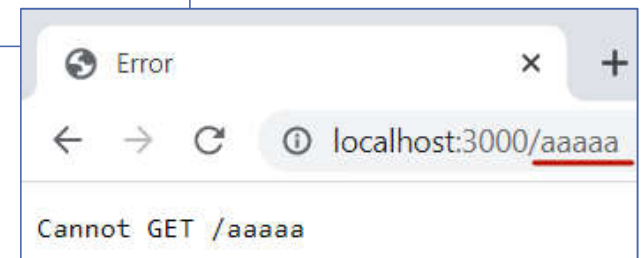
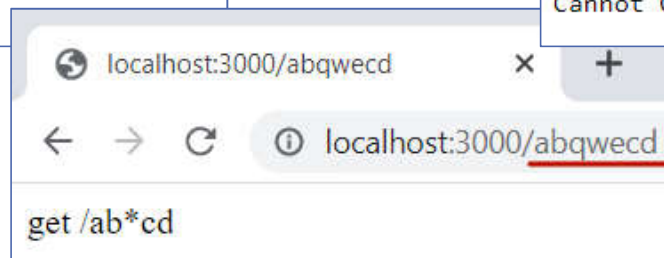
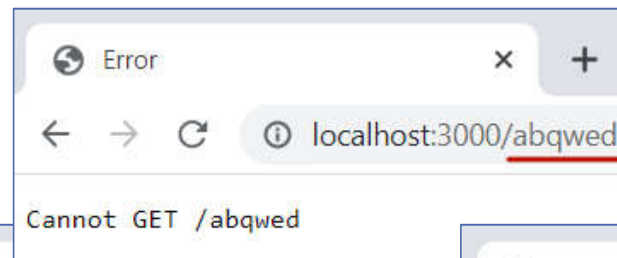
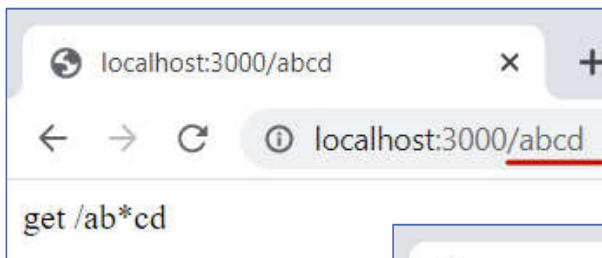
```
app.get('/X/*', (req, res, next) => {  
  res.send('get /X/*');  
});
```



Маршрутизация (путь – паттерн)

* – произвольный набор символов

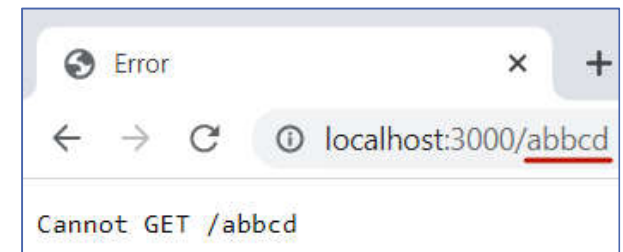
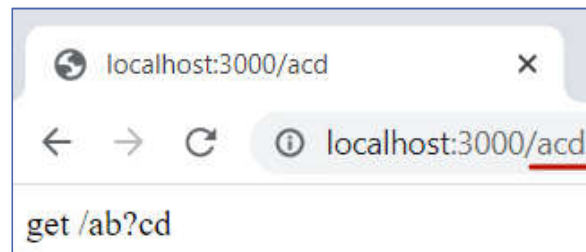
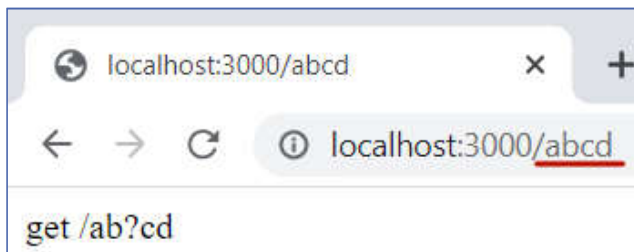
```
app.get('/ab*cd', (req, res, next) => {  
  res.send('get /ab*cd');  
});
```



Маршрутизация (путь – паттерн)

? – символ перед знаком ? может встречаться 0 или 1 раз

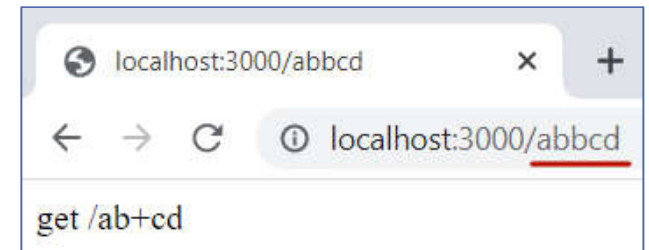
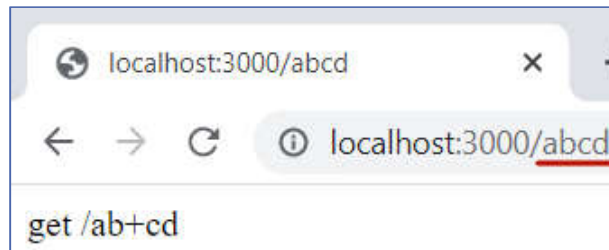
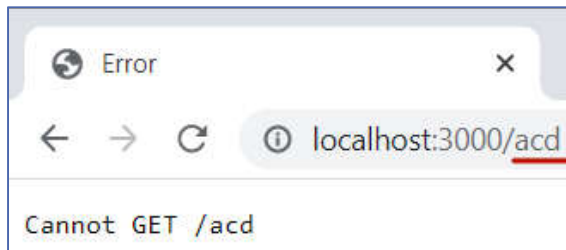
```
app.get('/ab?cd', (req, res, next) => {  
  res.send('get /ab?cd');  
});
```



Маршрутизация (путь – паттерн)

+ – символ перед знаком + может встречаться 1 и более раз

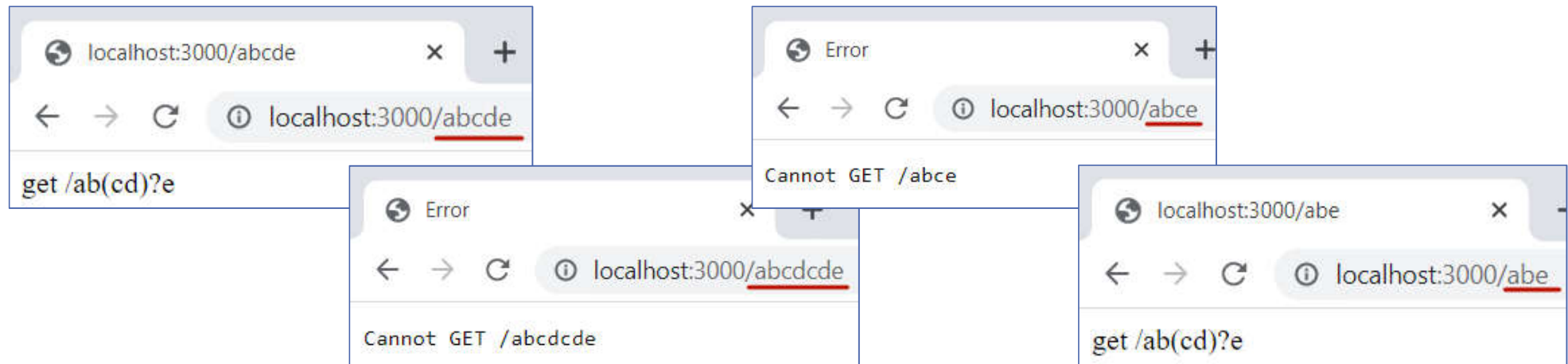
```
app.get('/ab+cd', (req, res, next) => {  
  res.send('get /ab+cd');  
});
```



Маршрутизация (путь – паттерн)

() – группировка символов

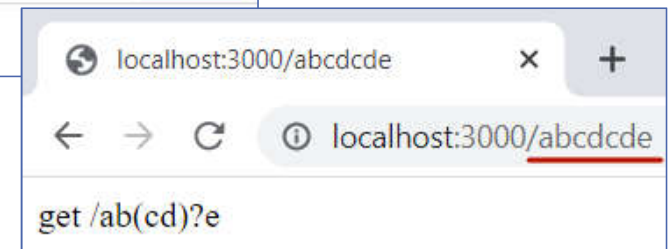
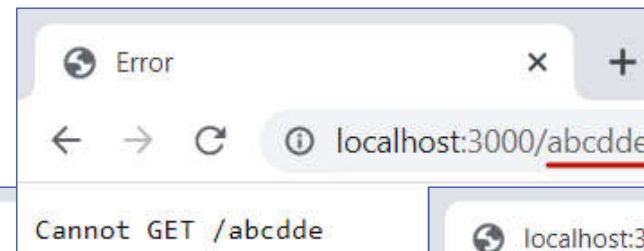
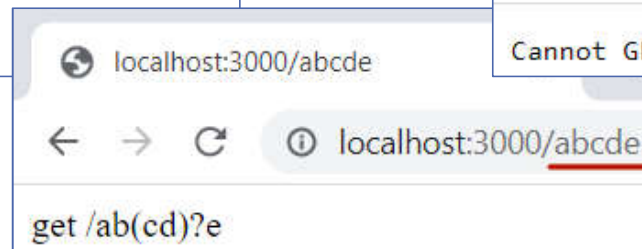
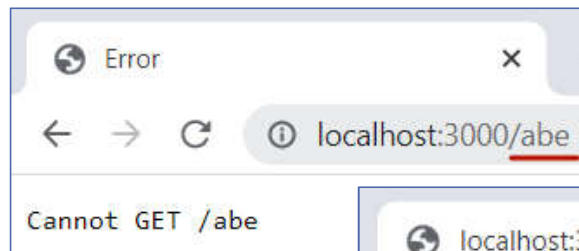
```
app.get('/ab(cd)?e', (req, res, next) => {  
  console.log('get /ab(cd)?e');  
  res.send('get /ab(cd)?e');  
});
```



Маршрутизация (путь – паттерн)

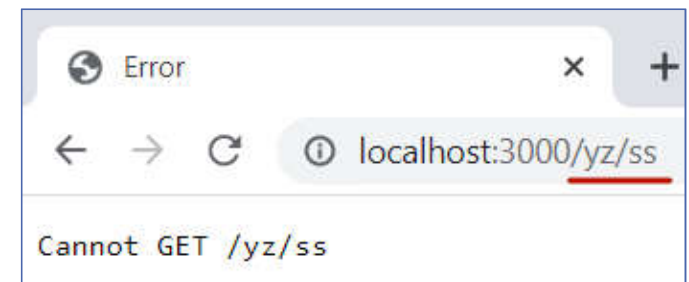
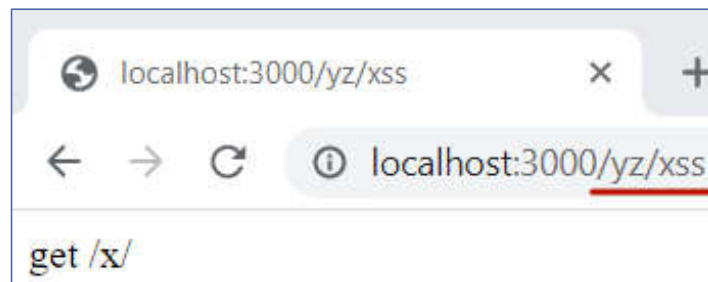
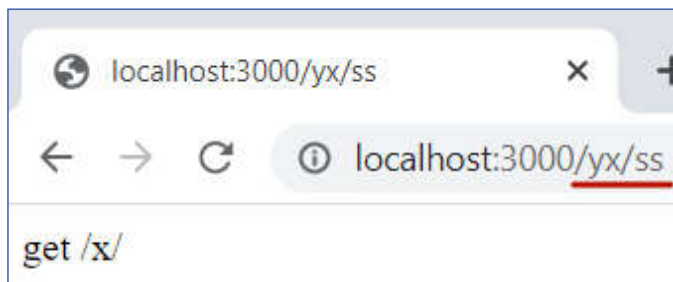
() – группировка символов

```
app.get('/ab(cd)+e', (req, res, next) => {  
  res.send('get /ab(cd)?e');  
});
```



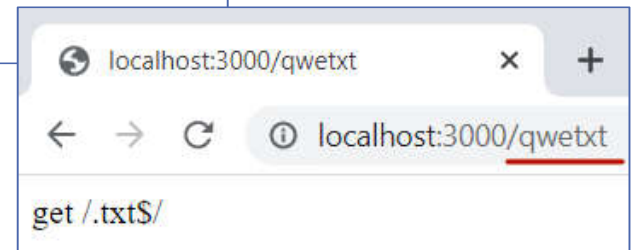
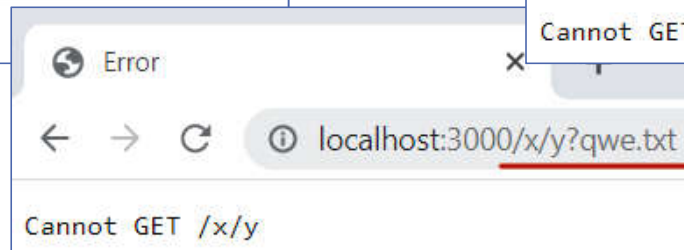
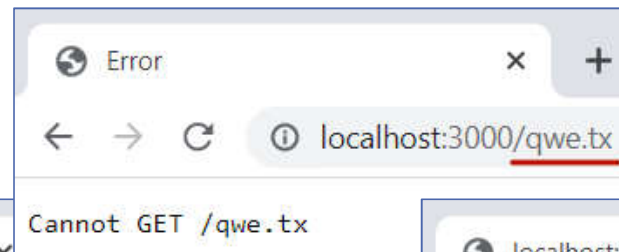
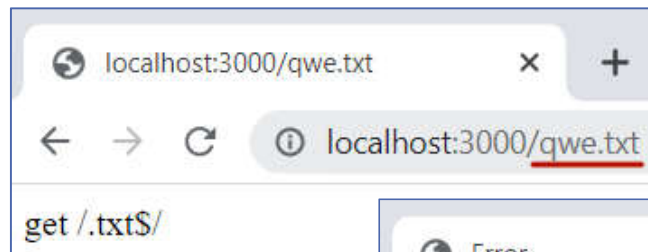
Маршрутизация (путь – регулярное выражение)

```
app.get(/x/, (req, res) => {  
  res.send('/x/')  
})
```



Маршрутизация (путь – регулярное выражение)

```
app.get(/.txt$/, (req, res) => {  
  res.send('get /.txt$/')  
})
```



Контейнер обработки запроса

```
const express = require("express"); // npm install express
const app = express();

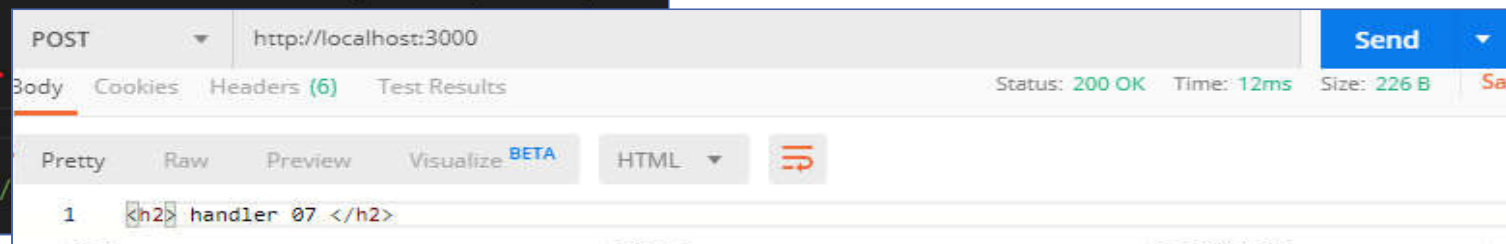
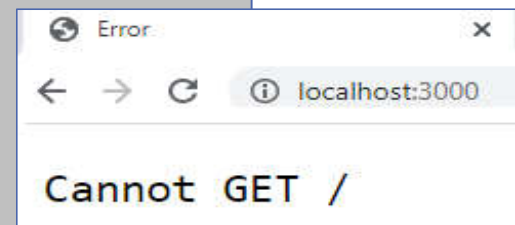
app.use((req, res, next)=>{console.log('handler 01'); next();}); // middleware
app.use((req, res, next)=>{console.log('handler 02'); next();}); // middleware
app.use((req, res, next)=>{console.log('handler 03'); next();}); // middleware

app.post('/', (req, res, next)=>{ // обработка post-запросов
  console.log('post / handler 05');
  next();
})
app.post('/', (req, res, next)=>{ // обработка post-запросов
  console.log('post / handler 06');
  next();
})
app.post('/', (req, res, next)=>{ // обработка post-запросов
  console.log('post / handler 07');
  next();
})
app.post('/', (req, res)=>{ // обработка post-запросов
  console.log('post / handler 08');
  res.send("<h2> handler 07 </h2>");
})

var server = app.listen(3000);
```

Для одного endpoint'а может быть несколько обработчиков. При наличии нескольких обработчиков важно **вызвать next()** в обработчике, **чтобы передать управление следующему обработчику в цепочке**, либо отправить клиенту.

```
D:\PSCA\Lec17_express>node 17-03
handler 01
handler 02
handler 03
handler 01
handler 02
handler 03
post / handler 05
post / handler 06
post / handler 07
post / handler 08
```



```

const express = require('express');           // npm install express
const app = express();

app.use((req, res, next) => {                  // middleware
  console.log('before-handler 01');
  next();
  console.log('after-handler 01');             // на обратном пути петли
});
app.get('/', (req, res, next) => {             // обработка get-запросов
  console.log('get / handler 02', req.query);
  next();
});
app.get('/', (req, res, next) => {             // обработка get-запросов
  console.log('get / handler 03');
  if(!req.query.x) res.send('<h2> handler 03 </h2>');
  else if(req.query.x == '0') res.send('<h2> handler 03 x = 0</h2>');
  else if(req.query.x == '1') throw 'error handler 03'; // генерация ошибки
  else next();
});
app.use((req, res, next) => {                  // middleware
  console.log('handler 04');
  next();
});
app.use((err, req, res, next) => {             // middleware: обработка ошибок
  console.log('error-handler', err);
  res.status(500).send('error handler')
});
app.use((req, res, next) => {                  // middleware: конечная точка
  console.log('notfound-handler');
  res.status(404).send('notfound-handler')
});

var server = app.listen(3000);                // = http.Server

```

Обратное движение по конвейеру

Method	Path	Response	Console
GET	/	before-handler 01 get / handler 02 {} get / handler 03 after-handler 01	<div>Body ▾  200 OK</div> <div> Pretty Raw Preview Visualize <div>HTML ▾ </div> </div> <div>1 <code><h2> handler 03 </h2></code></div>
GET	/?x=0	before-handler 01 get / handler 02 { x: '0' } get / handler 03 after-handler 01	<div>Body ▾  200 OK</div> <div> Pretty Raw Preview Visualize <div>HTML ▾ </div> </div> <div>1 <code><h2> handler 03 x = 0</h2></code></div>
GET	/?x=1	before-handler 01 get / handler 02 { x: '1' } get / handler 03 error-handler error handler 03 after-handler 01	<div>Body ▾  500 Internal Server Error</div> <div> Pretty Raw Preview Visualize <div>HTML ▾ </div> </div> <div>1 error handler</div>
GET	/?x=2	before-handler 01 get / handler 02 { x: '2' } get / handler 03 handler 04 notfound-handler after-handler 01	<div>Body ▾  404 Not Found</div> <div> Pretty Raw Preview Visualize <div>HTML ▾ </div> </div> <div>1 notfound-handler</div>
POST	/	before-handler 01 handler 04 notfound-handler after-handler 01	<div>Body ▾  404 Not Found</div> <div> Pretty Raw Preview Visualize <div>HTML ▾ </div> </div> <div>1 notfound-handler</div>

Остановка сервера

```
app.use((req, res, next)=>{           //middleware
  console.log('handler 02');
  next();
});

app.use((req, res, next)=>{           //middleware
  console.log('handler 03');
  res.send("<h2> handler 03 </h2>");
  server.close();
});

var server = app.listen(3000);       // = http.Server
```

app.listen() возвращает
экземпляр класса
http.Server.

Для остановки сервера
нужно вызывать close()
для этого экземпляра, а
не для экземпляра
приложения.

Свойства объекта request

- **req.app** – содержит ссылку на экземпляр приложения Express;
- **req.baseUrl** – путь URL, по которому был смонтирован экземпляр маршрутизатора;
- **req.method** – содержит строку, соответствующую методу HTTP-запроса;
- **req.ip** – содержит удаленный IP-адрес запроса;
- **req.originalUrl** – похоже на req.url; однако он сохраняет исходный URL-адрес запроса, что позволяет вам свободно переписывать req.url для внутренней маршрутизации;
- **req.protocol** – содержит строку протокола запроса: http или https;

Свойства объекта request

- **req.path** – содержит path из URL запроса;
- **req.hostname** – содержит имя хоста, полученное из заголовка HTTP;
- **req.params** – объект, содержащий свойства, сопоставленные с именованным route (через :);
- **req.query** – объект, содержащий свойство для каждого параметра строки запроса в маршруте. Если нет строки запроса, это пустой объект, {};
- **req.body** – содержит пары ключ-значение данных, представленных в теле запроса. По умолчанию оно пустое и заполняется при использовании промежуточных обработчиков для анализа тела;
- **req.route** – содержит текущий согласованный маршрут, строку;

Свойства объекта request

- **req.cookies** – объект, который содержит cookie, отправленные в запросе (заполняется при использовании промежуточного обработчика);
- **req.signedCookies** – содержит подписанные cookie, отправленные в запросе (заполняется при использовании промежуточного обработчика);
- **req.secure** – логическое свойство, которое имеет значение true, если установлено соединение TLS;
- **req.stale** – указывает, является ли ответ в кэше клиента «устаревшим»;
- **req.fresh** – указывает, является ли ответ в кэше клиента «свежим»;
- **req.subdomains** – массив поддоменов в доменном имени запроса;
- **req.xhr** – логическое свойство, которое имеет значение true, если запрос был выполнен клиентской библиотекой, такой как jQuery.

Метод req.accepts(types)

Проверяет, являются ли указанные типы контента **приемлемыми для клиента**, на основании заголовка запроса Accept.

```
app.post('/xxx', (req, res, next)=>{                                // обработка post-запросов

  console.log('json? = ', req.accepts('json'));
  console.log('html? = ', req.accepts('html'));
  console.log('[html, json]? = ', req.accepts(['json', 'html']));
  res.send('<h1>post</h1>');
})
```

Метод возвращает **наилучшее совпадение или**, если ни один из указанных типов содержимого не является приемлемым, возвращает **false**.

Метод req.accepts()

POST

▼

http://localhost:3000/xxx

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

▼ Headers (1)

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	Accept	application/json,text/html	

D:\PSCA\Lec17_express>node 17-05
Start server, port: 3000
json? = json
html? = html
[html, json]? = json

POST

▼

http://localhost:3000/xxx

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

▼ Headers (1)

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	Accept	image/png,text/html	

D:\PSCA\Lec17_express>node 17-05
Start server, port: 3000
json? = false
html? = html
png? = png
[html, json, png]? = png

Свойства объекта response

- **res.app** – содержит ссылку на экземпляр приложения Express, res.app идентичен свойству req.app;
- **res.headersSent** – логическое свойство, указывающее, отправляло ли приложение заголовки HTTP для ответа;
- **res.locals** – объект, который содержит локальные переменные ответа, ограниченные областью запроса.

Метод res.send([body])

```
app.get('/sendHtml', (req, res, next) => {  
  res.send('<p>some html</p>')  
})
```

Body ▾

Pretty Raw Preview Visualize

1 <p>some html</p>|

GET	http://localhost:3000/sendHtm	
Params Auth Headers (7) Body Pre-req. Tests Settings		
Headers ▾		🌐 200 OK 8 ms 244 B
Key	Value	
X-Powered-By	ⓘ	Express
Content-Type	ⓘ	text/html; charset=utf-8
Content-Length	ⓘ	16

Отправляет HTTP-ответ и автоматически назначает заголовки **Content-Length** и **Content-Type** (если они не определены ранее) .

Метод res.format(object)

```
app.post('/format', (req, res, next)=>{  
  res.format(  
    {  
      'text/html'      : ()=>{res.send('<h2>format: text/html</h2>')},  
      'text/plain'     : ()=>{res.send('format: text/plain')},  
      'application/json': ()=>{res.send('{"format":"application/json"}')},  
      'default'        : ()=>{res.status(406).send('Not Acceptable')}  
    }  
  );  
})
```

Выполняет согласование содержимого **на основе заголовка запроса Accept**.
В результате **устанавливается заголовок ответа Content-Type**.

Метод res.format()

POST http://localhost:3000/format Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Headers 9 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	Accept	text/html			
	Key	Value			

Body Cookies (1) Headers (7) Test Results

Pretty Raw Preview Visualize HTML

1 `<h2>format: text/html</h2>`

POST http://localhost:3000/format Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Headers 9 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	Accept	text/plain			
	Key	Value	Description		

Body Cookies (1) Headers (7) Test Results

Pretty Raw Preview Visualize Text

1 `format: text/plain`

Status: 200 OK Time: 25 ms Size: 238 B Save

POST http://localhost:3000/format

Params Authorization Headers (10) Body Pre-request Script Tests

Headers 9 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	Accept	image/png			
	Key	Value	Description		

Body Cookies (1) Headers (7) Test Results

Pretty Raw Preview Visualize HTML

1 `Not Acceptable`

Status: 406 Not Acceptable Time: 24 ms Size: 244 B Save

Метод res.append(field [, value])

```
app.post('/append', (req, res, next)=>{  
  res.append('X-xyz', 'x=3;y=4;z=5;');  // добавить заголовок  
  res.send('<h1>post/properties</h1>');  
})
```

Добавляет указанное значение в указанный заголовок ответа.

POST	http://localhost:3000/append	Se
body	Cookies	Headers (7)
Status: 200 OK Time: 20ms Size: 2		
KEY	VALUE	
X-Powered-By	Express	
X-xyz	x=3;y=4;z=5;	
Content-Type	text/html; charset=utf-8	

Метод `res.attachment([filename])`

```
app.get('/attachment', (req, res, next)=>{  
  res.attachment('Kaz.png');           // добавить заголовок  
  let rs = fs.ReadStream('./Kaz.png');  
  rs.pipe(res);  
})
```

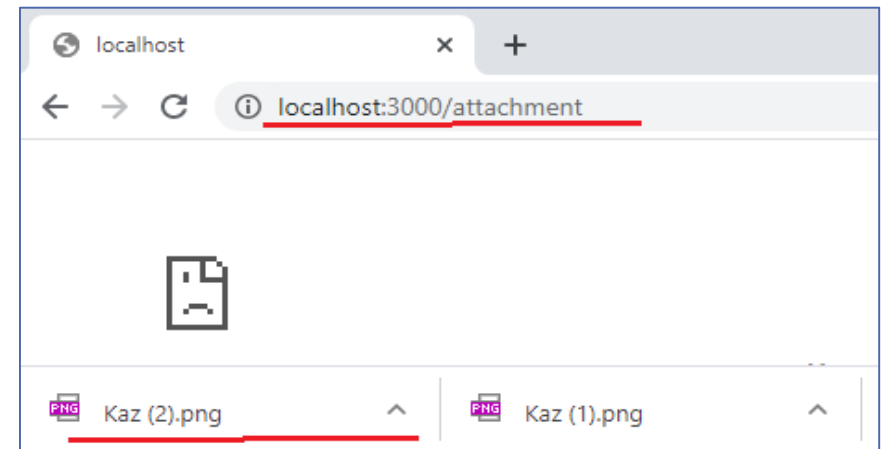
Устанавливает в заголовок ответа **Content-Disposition** значение «**attachment**». Если задано имя файла, тогда он устанавливает **Content-Type** на основе расширения файла и добавляет название в **Content-Disposition**.

GET ▼ http://localhost:3000/attachment

Body Cookies Headers (6) Test Results

Status: 200 OK Time: 17ms Size: 1.1 KB

KEY	VALUE
X-Powered-By ⓘ	Express
Content-Type ⓘ	image/png
Content-Disposition ⓘ	attachment; filename="Kaz.png"
Date ⓘ	Sat, 11 Jan 2020 22:52:30 GMT
Connection ⓘ	keep-alive



Заголовок Content-Disposition является индикатором того, что ожидаемый контент ответа будет отображаться в браузере, как веб-страница или часть веб-страницы, или же как вложение, которое затем может быть скачано и сохранено локально.

```
Content-Disposition: inline
```

```
Content-Disposition: attachment
```

```
Content-Disposition: attachment; filename="filename.jpg"
```

Метод `res.download(path [, filename] [, options] [, fn])`

```
app.get('/download', (req, res, next) => {
  res.download('./upload/kitty.jpg');
})
```

Устанавливает заголовки ответа **Content-Type** (на основе расширения файла) и **Content-Disposition** (значение attachment).

Этот метод использует `res.sendFile()` для передачи файла.

Метод res.download()

GET http://localhost:3000/download

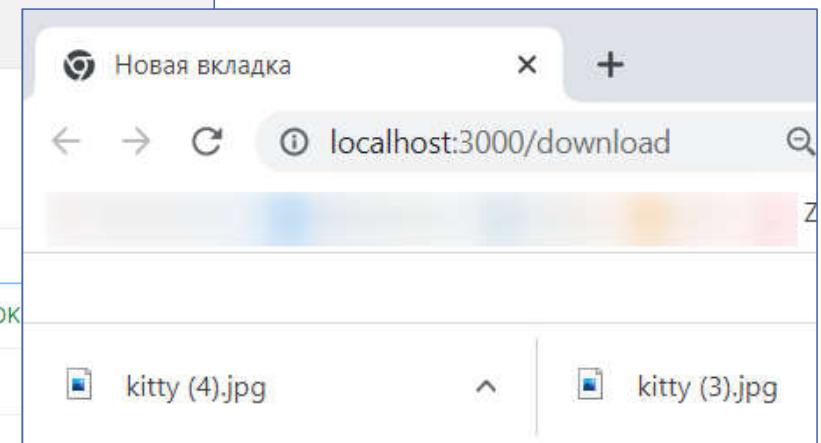
Params Authorization Headers (8) Body Pre-request Script Tests Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies (1) Headers (11) Test Results Status: 200 OK

Key	Value
X-Powered-By	Express
<u>Content-Disposition</u>	<u>attachment; filename="kitty.jpg"</u>
Accept-Ranges	bytes
Cache-Control	public, max-age=0
Last-Modified	Wed, 12 Apr 2023 19:22:47 GMT
ETag	W/"24047-18776eba64d"
<u>Content-Type</u>	<u>image/jpeg</u>



Метод `res.cookie(name, value [, options])`

```
app.post('/cookie', (req, res, next)=>{  
  res  
    .status(201)  
    .cookie('key-a', 'value-a')  
    .cookie('key-b', 'value-b')  
    .send('<h1>cookie<h2>');  
})
```

Устанавливает имя куки и значение. Параметр `value` может быть строкой или объектом, преобразованным в JSON.

Устанавливает заголовок ответа **Set-Cookie**, который используется для отправки cookies с сервера на агент пользователя.

Метод res.cookie()

The image shows two screenshots from a web browser's developer tools. The left screenshot shows the 'Application' tab with the 'Cookies' section expanded for the URL 'http://localhost:3000'. It displays two cookies: 'key-a' with value 'value-a' and 'key-b' with value 'value-b', both on the 'localhost' domain with a session expiration. The right screenshot shows a REST client (like Postman) with a POST request to 'http://localhost:3000/cookie'. The 'Headers' tab is selected, showing the response headers. Two 'Set-Cookie' headers are highlighted with red lines, corresponding to the cookies shown in the left screenshot: 'key-a=value-a; Path=/' and 'key-b=value-b; Path=/'.

Name	Value	Domain	Path	Expires / Max-Age	Size
key-a	value-a	localhost	/	Session	12
key-b	value-b	localhost	/	Session	12

KEY	VALUE
X-Powered-By	Express
Set-Cookie	key-a=value-a; Path=/
Set-Cookie	key-b=value-b; Path=/
Content-Type	text/html; charset=utf-8
Content-Length	14

Метод `res.clearCookie(name [, options])`

```
app.get('/clear-cookie', (req, res, next)=>{  
  
  res  
    .status(200)  
    .clearCookie('key-a')  
    .send('<h1>clear-cookie<h2>');  
  
})
```

Чтобы удалить cookie, сервер возвращает заголовок ответа **Set-Cookie** с датой истечения срока действия в прошлом.

GET	http://localhost:3000/clear-cookie	Send	Save
Body	Cookies (1)	Headers (7)	Test Results
Status: 200 OK Time: 19ms Size: 292 B Save Response			
KEY	VALUE		
X-Powered-By	Express		
Set-Cookie	key-a=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT		
Content-Type	text/html; charset=utf-8		
Content-Length	20		
ETag	W/"14-GqG8LdF3v79Ide/xw9N5v5m04Ig"		
Date	Sun, 12 Jan 2020 20:08:58 GMT		
Connection	keep-alive		

Метод res.location(path)

```
app.get('/location', (req, res, next)=>{  
  res.location('http://diskstation.belstu.by:5001');  
  res.send('location');  
})
```

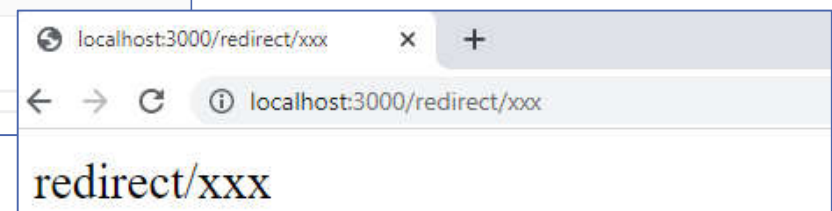
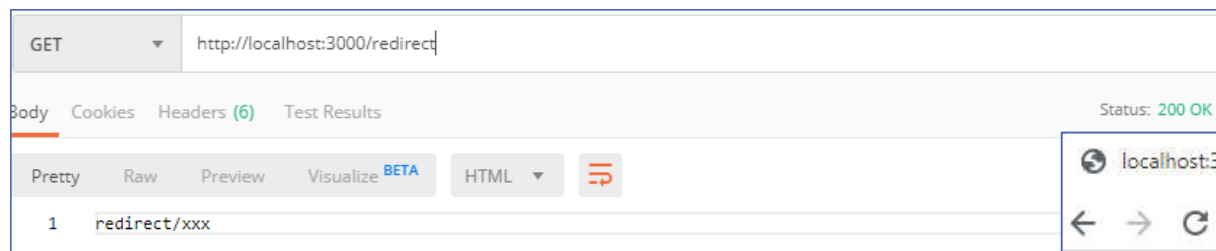
Устанавливает заголовок
ответа **Location** с указанным
параметром path.

GET	http://localhost:3000/location
KEY	VALUE
<input type="checkbox"/> Accept	image/png
Key	Value
► Temporary Headers (10) ⓘ	
Body	Cookies (1) Headers (7) Test Results
KEY	VALUE
X-Powered-By ⓘ	Express
Location ⓘ	http://diskstation.belstu.by:5001

Метод `res.redirect([status,] path)`

```
app.get('/redirect', (req, res, next)=>{  
  res.redirect('redirect/xxx');  
})  
app.get('/redirect/xxx', (req, res, next)=>{  
  res.send('redirect/xxx');  
})
```

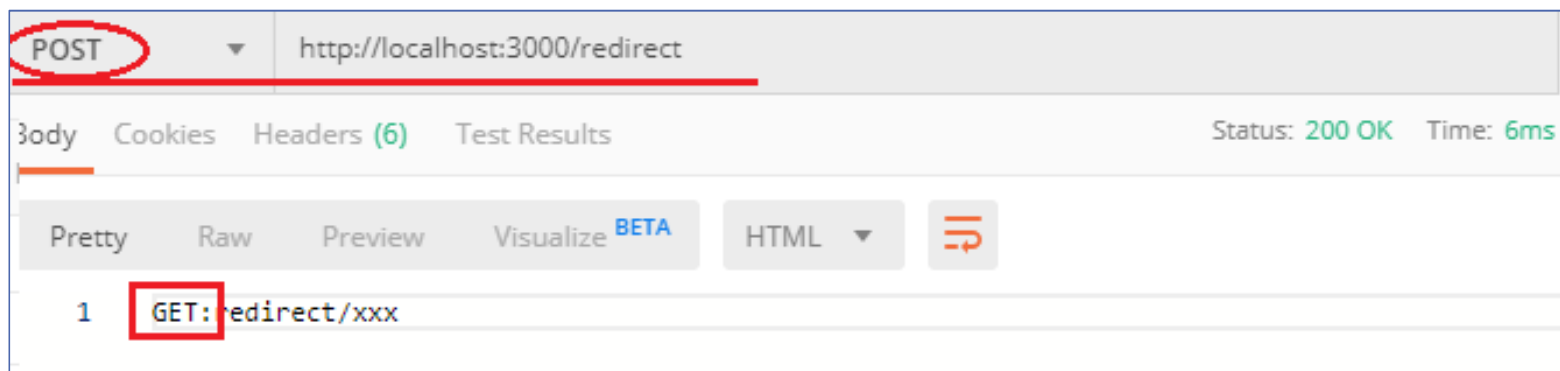
Переадресует на URL-адрес, полученный по указанному path (полный или относительный). Если статус код не указан, то по умолчанию отправляется 302 «Found».



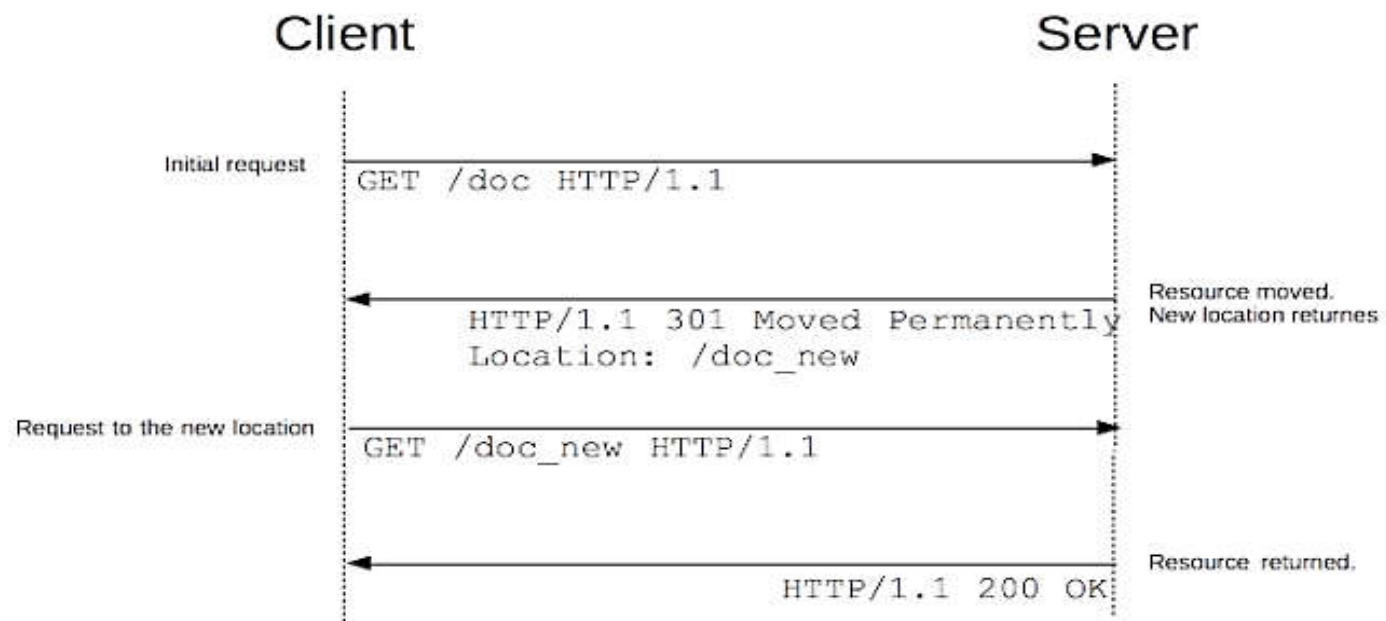
Метод res.redirect()

```
app.get('/redirect/xxx', (req, res, next)=>{  
  res.send('GET:redirect/xxx');  
})  
app.post('/redirect', (req, res, next)=>{  
  res.redirect('redirect/xxx');  
})  
app.post('/redirect/xxx', (req, res, next)=>{  
  res.send('POST:redirect/xxx');  
})
```

Если переадресовать POST-запрос на новый адрес, то на новый адрес будет послан GET-запрос из-за того, что браузер по историческим причинам посылает GET-запрос.



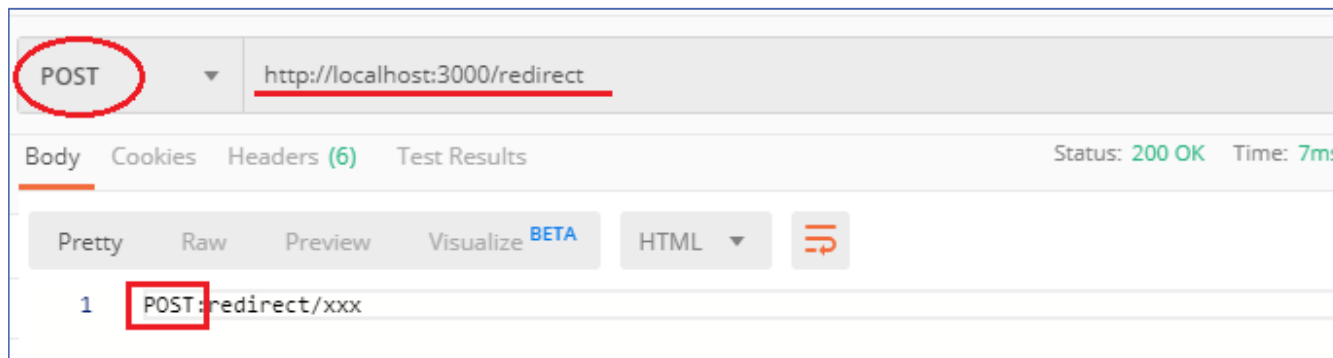
Переадресация



Метод res.redirect()

```
app.get('/redirect/xxx', (req, res, next)=>{
  res.send('GET:redirect/xxx');
})
app.post('/redirect', (req, res, next)=>{
  res.redirect(308, 'redirect/xxx');
})
app.post('/redirect/xxx', (req, res, next)=>{
  res.send('POST:redirect/xxx');
})
```

Если нам все же надо переадресовать именно **POST-запрос**, то в таком случае необходимо использовать **307 или 308 статус код**, т.к. они не позволяют изменить метод запроса с POST на GET.



Moved **Permanently**
301 => 308
Moved **Temporarily**
302 => 307

Метод `res.sendFile(path [, options] [, fn])`

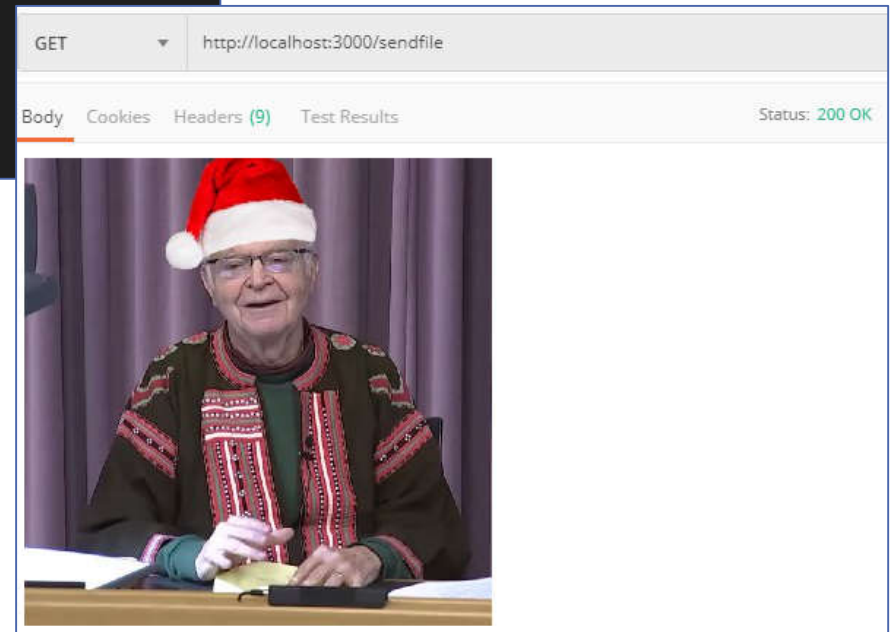
```
const path = require('path');

app.get('/sendfile', (req, res, next)=>{
  res.sendFile('DKnuth.jpg', {root: path.join(__dirname, 'Files')}, (err)=>{
    if(err) console.log(err);
    else console.log('send files - OK');
  });
});
```

Отправляет указанный **файл**.

Устанавливает заголовок **Content-Type** на основе расширения файла.

Если в объекте параметров не задан параметр `root`, путь должен быть абсолютным путем к файлу.

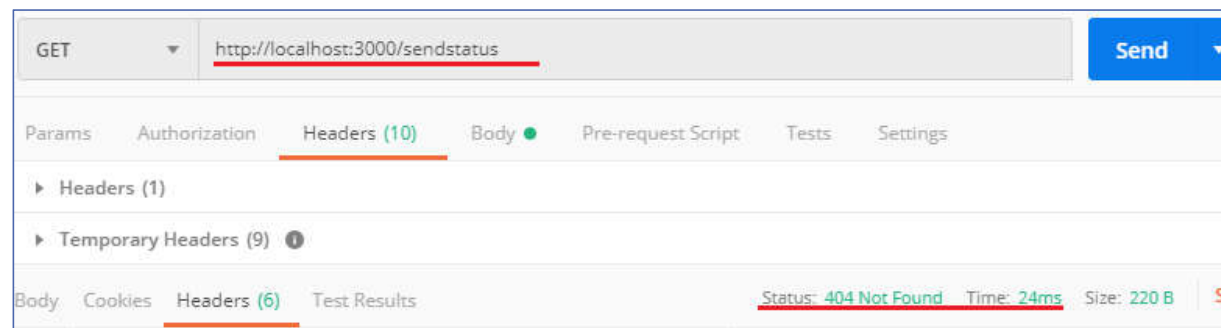


Метод `res.sendFile()`

- **`options.maxAge`** – устанавливает свойство `max-age` заголовка `Cache-Control` в миллисекундах
- **`options.root`** – корневой каталог для относительных имен файлов.
- **`options.lastModified`** – булевое свойство, устанавливает в заголовке `Last-Modified` дату последнего изменения файла в ОС. По умолчанию `true`.
- **`options.headers`** – объект, содержащий заголовки HTTP для использования с файлом.
- **`options.cacheControl`** – булевое свойство, включает или отключает настройку заголовка ответа `Cache-Control`. По умолчанию `true`.
- **`options.immutable`** – булевое свойство, включает или отключает директиву `immutable` в заголовке ответа `Cache-Control`. Директива `immutable` не позволяет клиентам делать условные запросы в течение срока действия опции `maxAge`, чтобы проверить, изменился ли файл. По умолчанию `false`.

Метод `res.sendStatus(statusCode)`

```
app.get('/sendstatus', (req, res, next)=>{  
  res.sendStatus(404);           //res.status(404).send('Not Found')  
})
```



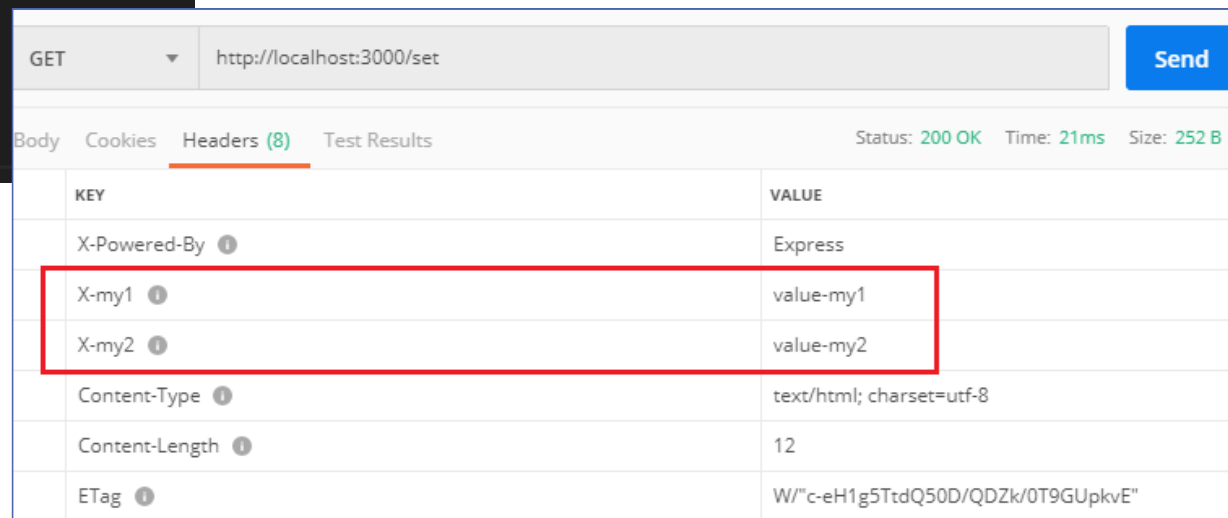
Устанавливает код ответа и отправляет его пояснение в качестве тела ответа (если указан неподдерживаемый код, то строковая версия кода).

Метод `res.set(field [, value])`

```
app.get('/set', (req, res, next)=>{  
  res.set(  
    {  
      'X-my1': 'value-my1',  
      'X-my2': 'value-my2'  
    }  
  );  
  res.send('--- SET ---');  
})
```

Устанавливает `value` в **заголовок** ответа `field`.

Чтобы установить несколько полей одновременно, передайте объект в качестве параметра.

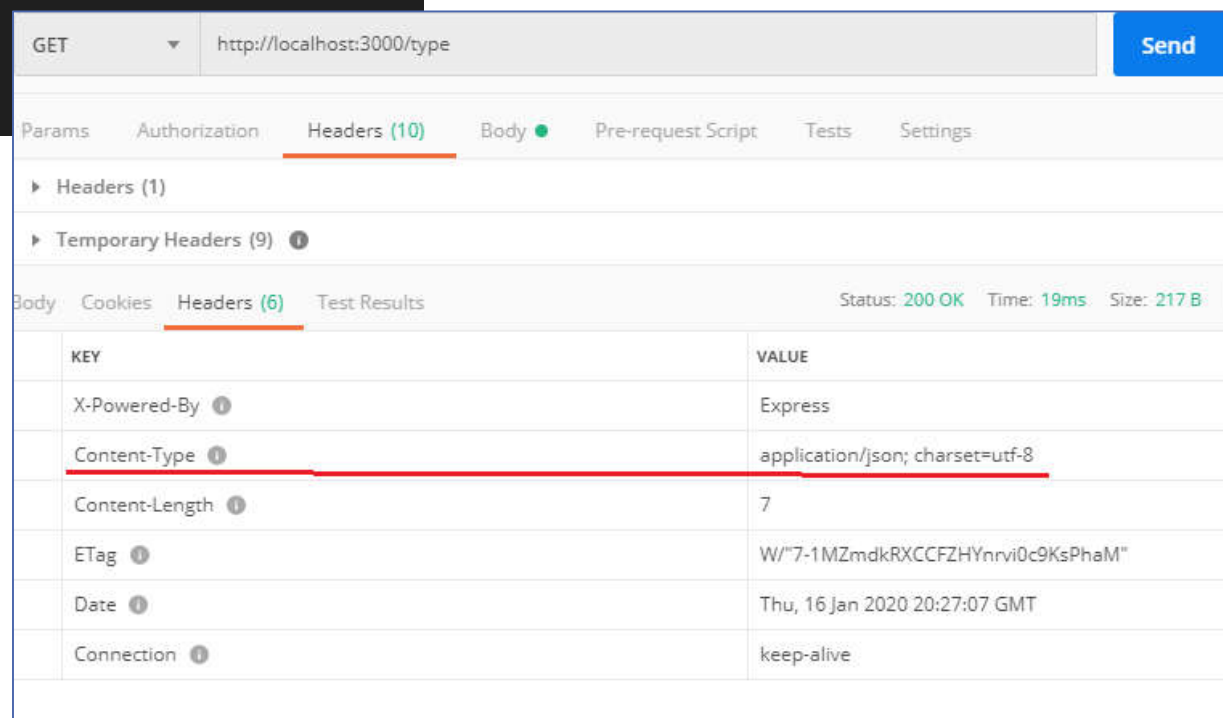


GET	http://localhost:3000/set	Send
Body	Cookies	Headers (8)
Status: 200 OK Time: 21ms Size: 252 B		
KEY	VALUE	
X-Powered-By	Express	
X-my1	value-my1	
X-my2	value-my2	
Content-Type	text/html; charset=utf-8	
Content-Length	12	
ETag	W/"c-eH1g5TtdQ50D/QDZk/0T9GUpkvE"	

Метод res.type(type)

```
app.get('/type', (req, res, next) => {  
  res.type('json');    // content-type  
  res.send({ x: 3 });  
})
```

Устанавливает для заголовка **Content-Type** тип MIME, определенный указанным type.



GET http://localhost:3000/type

Params Authorization Headers (10) Body Pre-request Script Tests Settings

► Headers (1)

► Temporary Headers (9)

Body Cookies Headers (6) Test Results Status: 200 OK Time: 19ms Size: 217 B

KEY	VALUE
X-Powered-By	Express
Content-Type	application/json; charset=utf-8
Content-Length	7
ETag	W/"7-1MZmdkRXCCFZHYnrvi0c9KsPhaM"
Date	Thu, 16 Jan 2020 20:27:07 GMT
Connection	keep-alive

Метод res.get(field)

```
app.get('/getType', (req, res, next) => {  
  res.type('html');  
  console.log(res.get('Content-Type'));  
  
  res.type('image');  
  console.log(res.get('Content-Type'));  
  
  res.type('json');  
  console.log(res.get('Content-Type'));  
  
  res.send({ x: 3 });  
})
```

```
PS D:\NodeJS\samples\cwp_17> node 17-13  
Server running at port 3000  
text/html; charset=utf-8  
application/octet-stream  
application/json; charset=utf-8
```

Возвращает указанный заголовок ответа.

Метод res.json(body)

```
app.get('/json', (req, res, next) => {  
  res.json({ x: 3 });  
})
```

Отправляет **ответ** в **формате JSON** и устанавливает правильный тип содержимого в **заголовок Content-Type**.

The screenshot shows the Chrome DevTools network panel for a GET request to `http://localhost:3000/json`. The response is a 200 OK status with a response time of 21 ms and a size of 240 B. The 'Headers' tab is selected, showing the following headers:

Key	Value
X-Powered-By	Express
Content-Type	application/json; charset=utf-8
Content-Length	7
ETag	W/"7-1MZmdkRXCCFZHYnrvi0c9KsPhaM"
Date	Thu, 10 Aug 2023 14:00:00 GMT
Connection	keep-alive
Keep-Alive	

The 'Body' tab is also visible, showing the response in JSON format:

```
{  
  "x": 3  
}
```

Обработка query-параметров

```
const express = require('express');
const app = express();

app.get('/sum', (req, res) => {
  console.log(req.query);
  let x = +req.query.x;
  let y = +req.query.y;

  res.send(`x + y = ${x + y}`);
})

app.listen(3000, () => console.log(`Server running at port 3000`));
```

Query-параметры запроса могут быть получены из свойства **query** в объекте **request** в обработчике на необходимый endpoint. Оно имеет форму объекта, в котором можно напрямую получить доступ к интересующим параметрам запроса.

Виды параметров запроса

- **Query-параметры** – дополнительная информация, которую можно добавить в URL-адрес. Состоит из двух обязательных элементов: самого параметра и его значения, разделенных знаком равенства (=). Параметры **указываются в конце URL**, отделяясь от основного адреса знаком вопроса (?). Можно указать более одного параметра, для этого каждый параметр со значениями отделяется от следующего знаком амперсанда (&).
- **Path-параметры** - дополнительная информация, которую можно задать в URL-адресе. Они **являются частью маршрута URL**.

/car/make/**12**/model?**color=mintgreen&doors=4**

Обработка query-параметров

GET

Params ☒ Auth Headers (7) Body Pre-req. Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	x	3
<input checked="" type="checkbox"/>	y	10

Body

Pretty Raw Preview Visualize HTML

GET

Params ☒ Auth Headers (7) Body Pre-req. Tests Settings

<input checked="" type="checkbox"/>	x	3
<input checked="" type="checkbox"/>	y	10
<input checked="" type="checkbox"/>	z	12

Body

Pretty Raw Preview Visualize HTML

GET

Params ☒ Auth Headers (7) Body Pre-req. Tests Settings

<input checked="" type="checkbox"/>	x	3
<input type="checkbox"/>	y	10
<input checked="" type="checkbox"/>	z	12

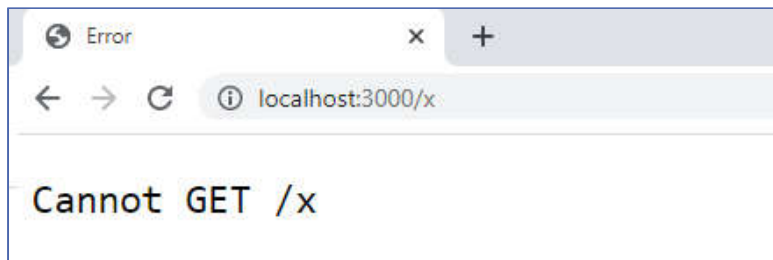
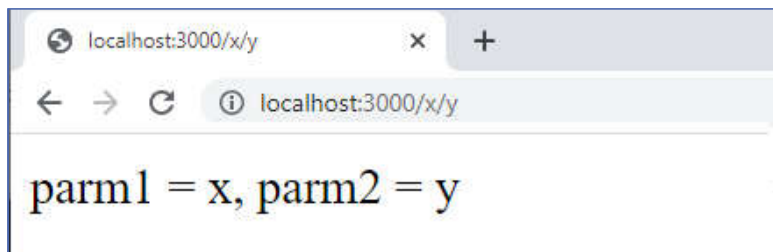
Body

Pretty Raw Preview Visualize HTML

```
PS D:\NodeJS\samples\cwp_17> node 17-14
Server running at port 3000
{ x: '3', y: '10' }
{ x: '3', y: '10', z: '12' }
{ x: '3', z: '12' }
```

Обработка path-параметров

```
app.get('/:parm1/:parm2', (req, res) => {  
  res.send(`parm1 = ${req.params.parm1}, parm2 = ${req.params.parm2}`);  
})
```



При определении маршрутов с динамически изменяющейся частью, необходимо **указать имя переменной-заполнителя** (начинающееся с **двоеточия**), и Express поместит эту переменную из входящего запроса в объект **req.params**.

Обработка тела (JSON)

Middleware `bodyparser` анализирует тело входящих запросов перед обработчиками и записывает результат разбора в свойство `req.body`.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

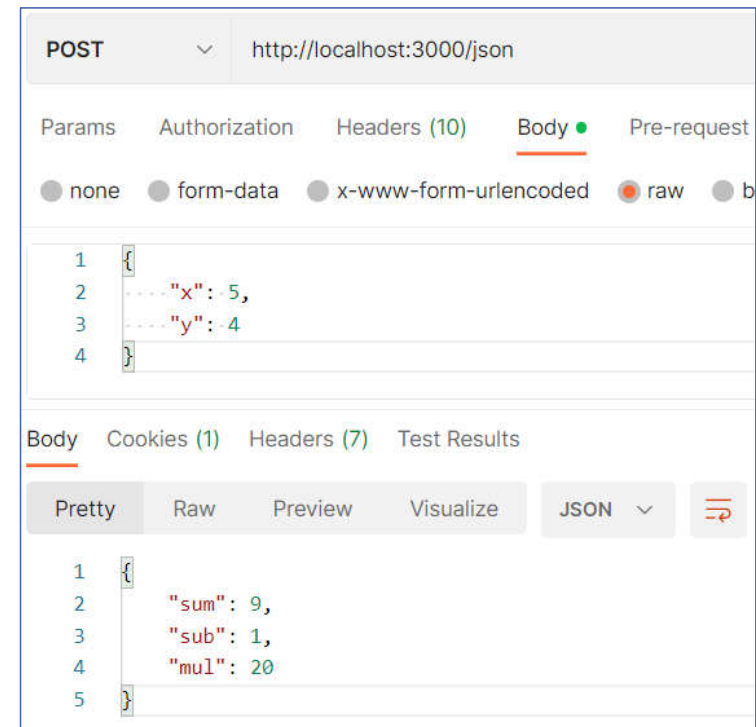
app.use(express.json());
// == app.use(bodyParser.json());

app.post('/json', (req, res) => {
  console.log(req.body);
  let x = +req.body.x;
  let y = +req.body.y;

  res.json({ sum: x + y, sub: x - y, mul: x * y });
})

app.listen(3000, () => console.log(`Server running at port 3000`));
```

В случае с `.json()` преобразует входящую строку из тела запроса в JS-объект. Обработывает только те запросы, в которых **Content-Type: application/json**.



```
PS D:\NodeJS\samples\cwp_17> node .\17-08.js
{ x: 5, y: 4 }
```


Обработка тела с формы (application/x-www-form-urlencoded)

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.use(express.urlencoded({ extended: false })); // querystring - false, qs - true
// == app.use(bodyParser.urlencoded({ extended: false }));

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/form09.html');
})

app.post('/', (req, res) => {
  console.log(req.body);
  res.send(`X: ${req.body.x} Y: ${req.body.y} S: ${req.body.s}`);
})

app.listen(3000, () => console.log(`Server running at port 3000`));
```

.urlencoded() анализирует только те запросы, в которых заголовок **Content-Type: application/x-www-form-urlencoded**.

Свойство req.body будет содержать пары ключ-значение, где значение может быть строкой или массивом (если extended - false) или любым типом (когда extended - true).

```
<!DOCTYPE html>
<html lang="en">

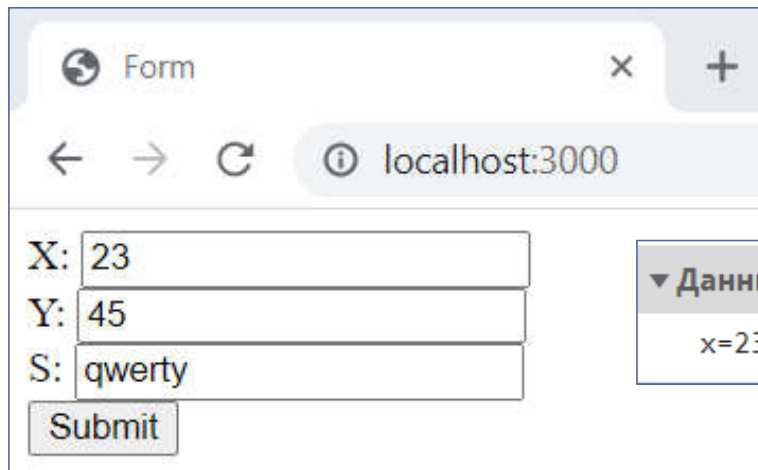
<head>
  <meta charset="UTF-8">
  <title>Form</title>
</head>

<body>
  <form method="post" action="/">
    X: <input type="number" name="x">
    <br>
    Y: <input type="number" name="y">
    <br>
    S: <input type="text" name="s">
    <br>
    <input type="submit" value="Submit">
  </form>
</body>

</html>
```

Обработка тела с формы (application/x-www-form-urlencoded)

Данные отправляются с формы по умолчанию в формате ключ-значение.
Ключ берется из атрибута **name** элемента формы, а значение из атрибута **value**.



Form

localhost:3000

X: 23

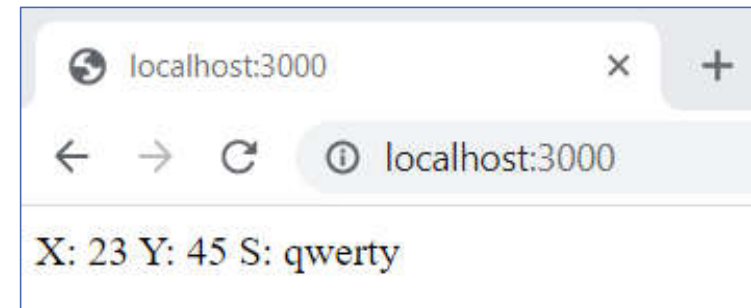
Y: 45

S: qwerty

Submit

▼ Данные форм посмотреть

x=23&y=45&s=qwerty



localhost:3000

X: 23 Y: 45 S: qwerty

```
PS D:\NodeJS\samples\cwp_17> node .\17-09.js
Server running at port 3000
[Object: null prototype] { x: '3', y: '5', s: 'ddd' }
```

```

const express = require('express');
const app = express();

const multer = require('multer');

const storageConfig = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'upload');
  },
  filename: (req, file, cb) => {
    cb(null, `${req.body.name}.${file.originalname.split('.')[1]}`);
  }
});

let upload = multer({ storage: storageConfig });

app.get('/upload', function (req, res, next) {
  res.sendFile(__dirname + '/form10.html');
});

app.post('/upload', upload.single('file'), async function (req, res, next) {
  if (!req.file)
    res.send('Ошибка при загрузке файла');
  else {
    res.send('Файл загружен');
  }
});

app.listen(3000, () => console.log(`Server listening on port 3000`));

```

Обработка тела с формы (multipart/form-data)

Multer – это сторонний middleware для обработки запросов с Content-Type: multipart/form-data, который в основном используется для загрузки файлов.

Multer **добавляет** объект **body** (текстовые поля формы) и объект **file** (или files) **внутри** объекта **request**.

Multer поставляется с двумя движками: DiskStorage и MemoryStorage, другие движки можно найти у сторонних разработчиков.

Обработка тела с формы (multipart/form-data)

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Upload</title>
</head>

<body>
  <form method="post" action="/upload" enctype="multipart/form-data">
    New name: <input type="text" name="name">
    <br>
    File: <input type="file" name="file">
    <br>
    <input type="submit" value="Submit">
  </form>
</body>

</html>
```

Upload

localhost:3000/upload

New name:

File:



localhost:3000/upload

localhost:3000/upload

Файл загружен

`express.Router` =

это изолированный экземпляр промежуточного ПО (middleware), позволяющий определить **дочерние подмаршруты** со своими обработчиками относительно некоторого главного маршрута. И тем самым связать подобный функционал в одно целое и упростить управление им.

express.Router

```
JS 17-07-router123.js > ...
1  const express = require('express');
2  const router = express.Router();
3
4  router.get('/1', (req, res) => { res.send('/1'); })
5  router.get('/2', (req, res) => { res.send('/2'); })
6  router.post('/3', (req, res) => { res.send('/3'); })
7
8  module.exports = router
```

```
JS 17-07-routerXYZ.js > ...
1  const express = require('express');
2  const router = express.Router();
3
4  router.get('/a', (req, res) => { res.send('/a'); })
5  router.get('/b', (req, res) => { res.send('/b'); })
6  router.put('/b', (req, res) => { res.send('/b'); })
7
8  module.exports = router
```

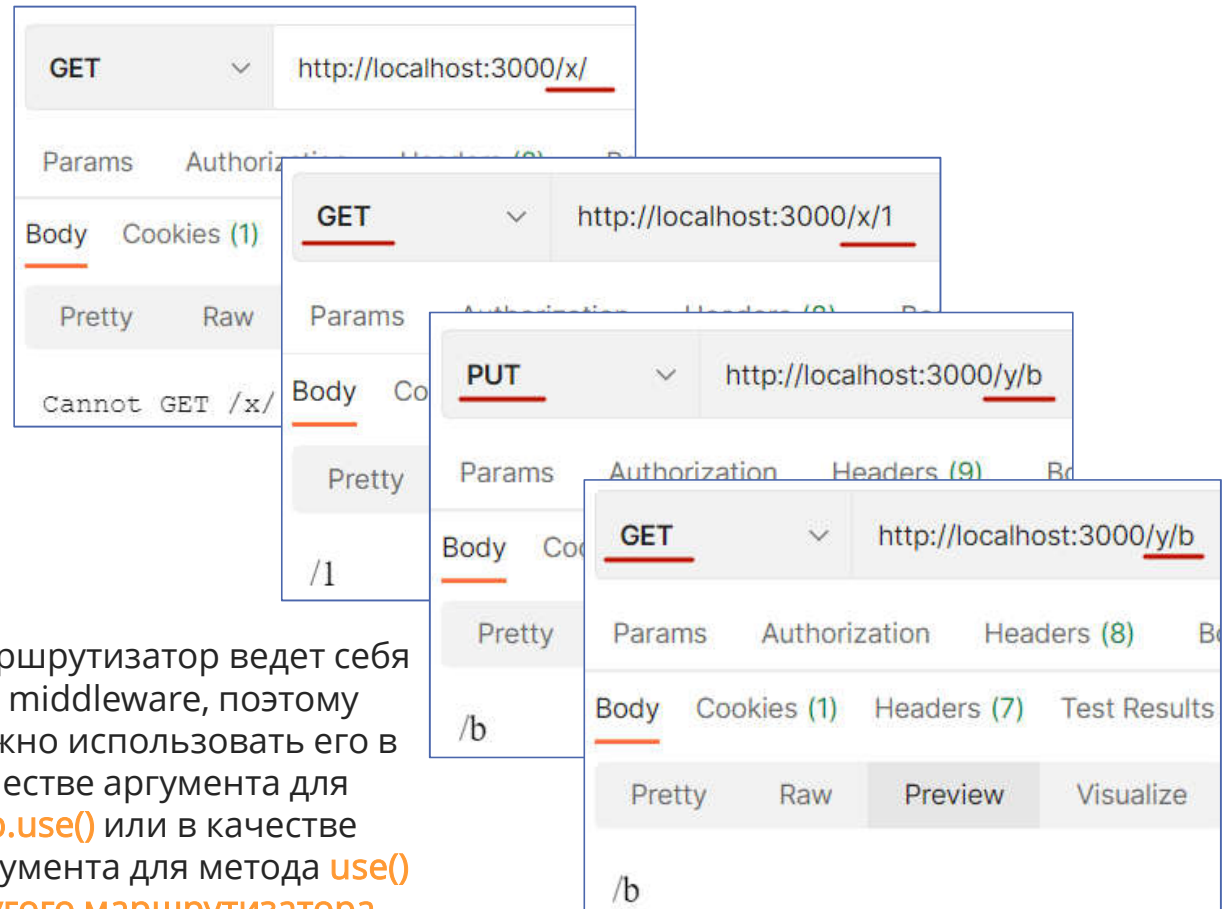
```
const express = require('express');
let app = express();

const router123 = require('./17-07-router123');
const routerXYZ = require('./17-07-routerXYZ');

app.use('/x', router123);
app.use('/y', routerXYZ);

app.listen(3000);
```

Маршрутизатор ведет себя как middleware, поэтому можно использовать его в качестве аргумента для `app.use()` или в качестве аргумента для метода `use()` другого маршрутизатора.



express.Router

```
const express = require('express');
const router = express.Router();

// router.get('/a', (req, res) => { res.send('/a'); })
// router.get('/b', (req, res) => { res.send('/b'); })
// router.put('/b', (req, res) => { res.send('/b'); })

router.get('/a', (req, res) => { res.send('/a'); })
router.route('/b')
  .get((req, res) => { res.send('/b'); })
  .put((req, res) => { res.send('/b'); })

module.exports = router
```

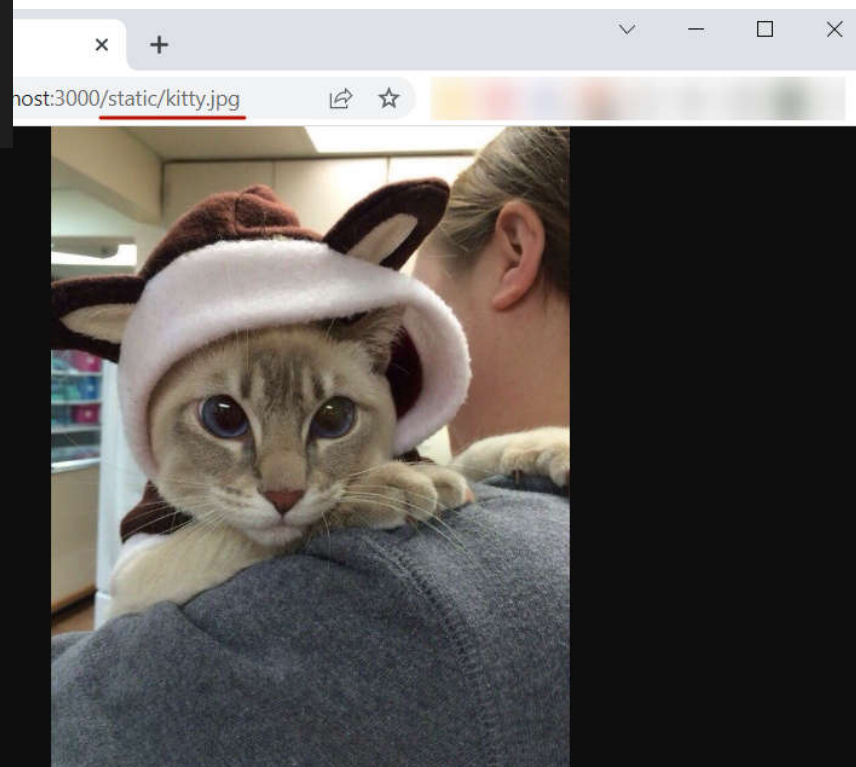
Можно использовать `router.route()`, чтобы избежать дублирования имен маршрутов.

Раздача статики

```
const express = require('express');  
const app = express();  
  
app.use('/static', express.static('./upload'));  
  
app.listen(3000, () => console.log(`Server running at port 3000`));
```

Статические файлы – это файлы расположенные **на стороне сервера** и предназначенные для **считывания** их **без изменения** с помощью HTTP GET-запроса по имени ресурса, включающего имя файла. Например, их можно получить с помощью тегов `<link>`, `<script>`, ``, `<audio>`, `<video>`, `<a>`, `<form>`, `<frame>`.

Для раздачи статических файлов в Express есть **middleware `express.static()`**, который указывает на каталог с файлами.



Cookies

HTTP headers | Set-Cookie

The **HTTP header Set-Cookie** is a response header and used to send cookies from the server to the user agent. So the user agent can send them back to the server later so the server can detect the user.

Syntax:

```
Set-Cookie: <cookie-name>=<cookie-value> | Expires=<date>  
           | Max-Age=<non-zero-digit> | Domain=<domain-value>  
           | Path=<path-value> | SameSite=Strict|Lax|none
```

Note: Using multiple directives are also possible.

```
HTTP/1.0 200 OK  
Content-type: text/html  
Set-Cookie: yummy_cookie=choco  
Set-Cookie: tasty_cookie=strawberry
```

```
GET /sample_page.html HTTP/1.1  
Host: www.example.org  
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

Cookie бывают **постоянные** и **сессионные**.

Возможные опции:

- **domain** – привязка cookie к поддомену;
- **path** – путь, на который распространяется действие cookie;
- **maxAge** – время жизни cookie в миллисекундах.
- **Expires** – дата истечения жизни cookie;
- **secure** – может применяться только с HTTPS;
- **httpOnly** – если true, то cookie не доступны через JavaScript;
- **SameSite** (Strict|Lax|none) – обеспечивает защиту от CSRF-атак, контроль отправки cookie на другой домен.

Работа с cookies

```
const express = require("express");
const app = express();
const cookieparser = require('cookie-parser')(); // npm install cookie-parser

app.use(cookieparser);

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

  let myid = req.cookies.myid;

  if (isFinite(myid)) ++myid;
  else myid = 0;

  res.cookie('myid', myid).send(`myid = ${myid}`);

});

app.listen(3000, () => console.log('Start server, port:', 3000));
```

Опции с предыдущего слайда можно указать в `res.cookie(name, value [, options])` в объекте третьим параметром.

```
D:\PSCA\Lec20_cookie>node 20-03
Start server, port: 3000
myid = 1
myid = 2
myid = 3
myid = 4
myid = 5
```

`res.cookie (name, value [, options])` устанавливает имя cookie и значение у клиента путем отправки заголовка ответа `Set-Cookie`.

`cookie-parser` – это middleware, который **анализирует cookie**, прикрепленные к запросу клиента в заголовке `Cookie` и **заполняет `req.cookies`** объектом, ключами которого являются имена cookie.

Работа с cookies (результат)

GET	http://localhost:3000/	Send
Postman-Token	8e095233-6ed2-4e5b-b87f-bb48a0e3c7d9	
Host	localhost:3000	
Accept-Encoding	gzip, deflate, br	
Content-Length	86	
Cookie	myid=8	
Connection	keep-alive	
Body Cookies (1) Headers (7) Test Results Status: 200 OK Time: 6ms Size: 239 B Sav		
KEY	VALUE	
X-Powered-By	Express	
Set-Cookie	myid=9; Path=/	
Content-Type	text/html; charset=utf-8	
Content-Length	8	
ETag	W/"8-tG6fPUoARVPr6pkr1IU\$PrMzN+E"	
Date	Fri, 13 Mar 2020 21:08:47 GMT	
Connection	keep-alive	

Работа с signed cookies

```
const express = require("express");
const app = express();
const cookieparser = require('cookie-parser'); // npm install cookie-parser

const cookiesecret = '1234567890';

app.use(cookieparser(cookiesecret));

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

  let myid = req.signedCookies.myid;

  if (isFinite(myid)) ++myid;
  else myid = 0;
  console.log('myid = ', myid);
  res.cookie('myid', myid, {signed:true}).send(`myid = ${myid}`);

});

app.listen(3000, () => console.log('Start server, port:', 3000));
```

Для подписи cookie первым параметром в cookieParser() нужно **передать строку или массив**.

Для того, чтобы создать подписанную cookie, надо в res.cookie() третьим параметром в объекте **указать свойство signed: true**.

```
D:\PSCA\Lec20_cookie>node 20-03
Start server, port: 3000
myid = 1
myid = 2
myid = 3
myid = 4
myid = 5
```

Работа с signed cookies (результат)

Подписанные cookie не отправляются в виде простого текста, они **кодируются в base64**. Эти данные не являются секретными, но они не могут быть подделаны, потому что **вторая часть cookie является подписью**.

GET

▼

http://localhost:3000/

Send

KEY	VALUE
Content-Type	text/plain
User-Agent	PostmanRuntime/7.22.0
Accept	*/*
Cache-Control	no-cache
Postman-Token	eef72e83-aa76-4d4a-b7bc-35297f95ee7b
Host	localhost:3000
Accept-Encoding	gzip, deflate, br
Content-Length	86
Cookie	myid=s%3A4.Hr92o0oDfYsaYl5nElkBUt7u4sXWAbTDiS%2B%2Bk4lbst0
Connection	keep-alive

Body

Cookies (1)

Headers (7)

Test Results

Status: 200 OK

Time: 8ms

Size: 289 B

Save

KEY	VALUE
X-Powered-By	Express
Set-Cookie	myid=s%3A5.djogYqkAz%2FpkolUpsYP3JaxQcUXXV6etCYJfw7SahXg; Path
Content-Type	text/html; charset=utf-8
Content-Length	8
ETag	W/"8-PU7fb/eJn8wLPnC1cRBQeX9Ye+U"
Date	Fri, 13 Mar 2020 21:42:05 GMT
Connection	keep-alive

Удаление cookie

```
const express = require("express");
const app = express();
const cookieparser = require('cookie-parser')(); // npm install cookie-parser

app.use(cookieparser);

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

  let myid = req.cookies.myid;

  if (isFinite(myid)) ++myid;
  else myid = 0;

  if (myid > 5) res.clearCookie('myid').send(`myid = ${myid=0}`);
  else res.cookie('myid', myid).send(`myid = ${myid}`);

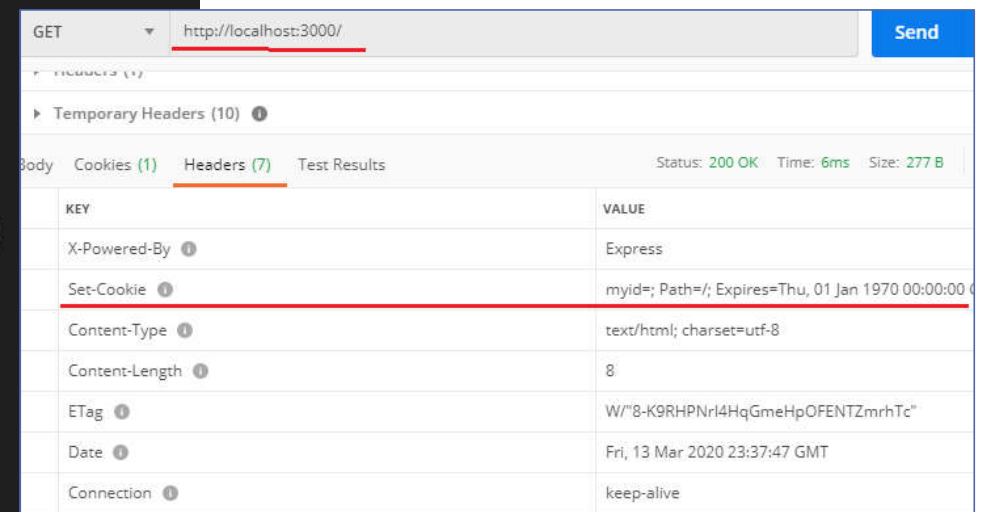
  console.log('myid = ', myid);

});

app.listen(3000, () => console.log('Start server, port:', 3000));
```

`res.clearCookie (name [, options])`

Удаление происходит путем установки опции Expires в 1 января 1970 г.



KEY	VALUE
X-Powered-By	Express
Set-Cookie	myid=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT
Content-Type	text/html; charset=utf-8
Content-Length	8
ETag	W/"8-K9RHPNrl4HqGmeHpOFENTZmrhTc"
Date	Fri, 13 Mar 2020 23:37:47 GMT
Connection	keep-alive

Работа с session

```
const app = express();
const cookiesecret = '1234567890';
const session = require('express-session')({
  resave: false,           // пересохранять ли, если нет изм.
  saveUninitialized: false, // сохранять ли пустые
  secret: cookiesecret,    // для подписи

  // store =      тип хранилища (Mem, DB)
  // cookie =     path, domain, secure, ...
});

app.use(session);

app.get('/', (req, res, next) => {
  if (!isFinite(req.session.mysesval)) req.session.mysesval = 0;
  else req.session.mysesval++;
  console.log('mysesval = ', req.session.mysesval);
  res.send(`mysesval = ${req.session.mysesval}`);
});

app.listen(3000, () => { console.log('Start server, port: ', 3000) });
```

express-session – middleware, который **создает session**, **устанавливает в cookie sessionId** и **создает** объект **req.session**.
Всякий раз, когда происходит запрос от того же клиента, по sessionId из заголовка Cookie будет происходить поиск соответствующей сессии и ее запись в req.session (учитывая, что сервер не был перезапущен).

```
D:\PSCA\Lec20_cookie>node 20-04
Start server, port: 3000
mysesval = 0
mysesval = 1
mysesval = 2
mysesval = 3
mysesval = 4
```


Работа с session (результат)

GET http://localhost:3000/ Send

Params Authorization Headers (11) Body Pre-request Script Tests Settings

► Headers (1)

▼ Temporary Headers (10) ⓘ

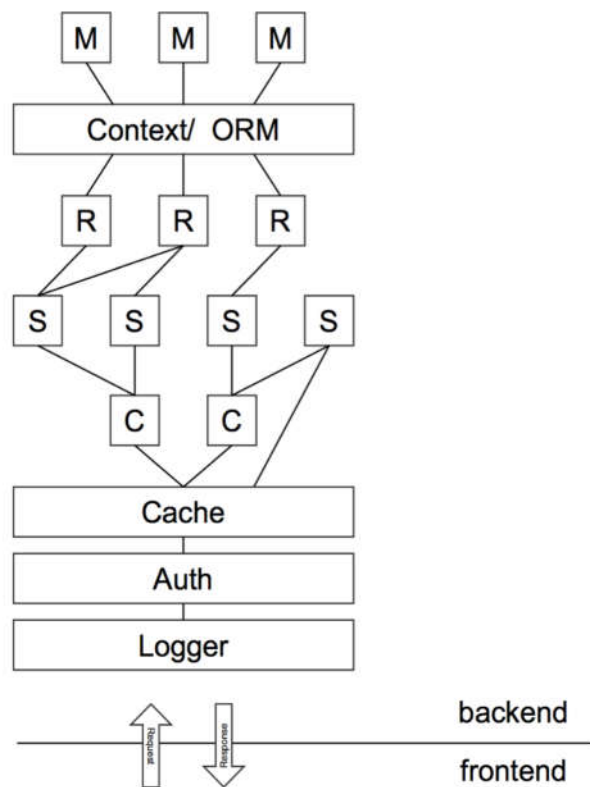
KEY	VALUE
Content-Type	text/plain
User-Agent	PostmanRuntime/7.23.0
Accept	*/*
Cache-Control	no-cache
Postman-Token	44cb7875-1242-462d-8e9a-88f050ff47de
Host	localhost:3000
Accept-Encoding	gzip, deflate, br
Content-Length	86
Cookie	connect.sid=s%3ASdkjz-_JR8oguVI3fBpgqhrE5kNNrSQN.o6jTmYbAurWpwt5tj
Connection	keep-alive

Body Cookies (1) Headers (6) Test Results Status: 200 OK Time: 8ms Size: 216 B Sav

Pretty Raw Preview Visualize HTML

1 mysesval = 4

Архитектура N-layer (или N-tier)



Контроллеры: обработчики маршрутов, которые разбирают входящие запросы, потребляет 1 и более сервисов.

Сервисы: организация бизнес-логики, которая может быть функциональной или объектно-ориентированной по своей природе, валидация, потребляет 1 и более репозитория, может возвращать DTO.

Репозитории: уровень доступа к данным, отвечает за выполнение запросов к базам данных, каждой модели – репозиторий, CRUD.

Контекст: подключение к бд, определение связей между моделями, перевод вызова методов в запросы, маппинг ответом на объекты

Модель: отражает сущность предметной области (таблица в БД)

- ✓ context
 - JS index.js
- ✓ controllers
 - JS facultyController.js
- ✓ global-controllers
 - JS errorController.js
- ✓ helpers
 - JS errors.js
 - JS validator.js
- ✓ models
 - JS faculty.js
- ✓ routers
 - JS facultyRouter.js
- ✓ services
 - JS facultyService.js
- JS app.js
- { } config.json

Структура проекта

Models

```
models > JS faculty.js > ...  
1  module.exports = (Sequelize, sequelize) => {  
2    ...  
3    return sequelize.define('Faculty', {  
4      faculty: { type: Sequelize.STRING, allowNull: false, primaryKey: true },  
5      faculty_name: { type: Sequelize.STRING, allowNull: false }  
6    }, {  
7      sequelize,  
8      tableName: 'Faculty',  
9      timestamps: false  
10   });  
11 }
```

Context

```
context > JS index.js > ...
1  const Sequelize = require('sequelize');
2  const config = require('../config.json')
3
4  const sequelize = new Sequelize(config.db.name, config.db.user, config.db.password, config.db.options);
5
6  const Faculty = require('../models/faculty')(Sequelize, sequelize)
7
8  // relations
9
10 module.exports = {
11   faculties: Faculty,
12
13   sequelize,
14   Sequelize,
15 };
16
```

```
{} config.json > ...
1  {
2    "db": {
3      "name": "SeqTest",
4      "user": "sa",
5      "password": "123",
6      "options": {
7        "host": "127.0.0.1",
8        "dialect": "mssql",
9        "dialectOptions": {
10         "options": {
11           "encrypt": false
12         }
13       }
14     }
15   }
16
17
```

Services

```
services > JS facultyService.js > ...
1  const Faculty = require('../context').faculties;
2  const errors = require('../helpers/errors');
3  const validator = require('../helpers/validator');
4
5  module.exports = {
6    getAll: async () => {
7      return Faculty.findAll();
8    },
9
10   getById: async (id) => {
11     let faculty = await Faculty.findOne({ where: { faculty: id } })
12     if (!faculty) throw errors.entityNotFound
13
14     return faculty;
15   },
16
17   addFaculty: async (data) => {
18     const validationResult = validator.check('addFaculty', data);
19     let faculty = await Faculty.findOne({ where: { faculty: data.faculty } });
20
21     if (faculty) throw errors.invalidId
22     if (validationResult.error) throw errors.invalidInput(validationResult.error.details[0].message)
23
24     return await Faculty.create(data);
25   }
26 }
```

Helpers (validation, errors)

```
helpers > JS validator.js > ...
1  const Joi = require('joi');
2
3  const schemas = {
4    'addFaculty': Joi.object().keys({
5      faculty: Joi.string().min(2).max(10).required(),
6      faculty_name: Joi.string().required()
7    })
8  };
9
10 exports.check = function (schema, body) {
11   if (!schemas[schema]) return {};
12
13   return schemas[schema].validate(body);
14 };
```

```
helpers > JS errors.js > ...
1  const express = require('express');
2
3  express.response.error = function (error) {
4    if (!error.code) {
5      error = {
6        message: error.toString(),
7        code: 'server_error',
8        status: 500
9      };
10   }
11
12   this.status(error.status).json(error);
13 };
14
15 module.exports = {
16   invalidId: {
17     message: 'Already exists',
18     code: 'already_exists',
19     status: 400
20   },
21   invalidInput: (message) => {
22     return {
23       message: message,
24       code: 'invalid_input',
25       status: 400
26     };
27   },
28   entityNotFound: {
29     message: 'Entity not found',
30     code: 'entity_not_found',
31     status: 404
32   },
33   methodNotAllowed: {
34     message: 'Method not allowed',
35     code: 'method_not_allowed',
36     status: 405
37   },
38   resourceNotFound: {
39     message: 'Resource not found',
40     code: 'resource_not_found',
41     status: 404
42   }
43 };
44
```

Controllers

```
global-controllers > JS errorController.js > ...  
1 module.exports = (error, req, res, next) => {  
2   // TODO: log error + 'res.locals.trace'  
3   res.error(error);  
4 };
```

```
controllers > JS facultyController.js > ...  
1 const facultyService = require('../services/facultyService');  
2  
3 module.exports = {  
4   getAll: async (req, res, next) => {  
5     try {  
6       res.json(await facultyService.getAll());  
7     } catch (error) {  
8       next(error)  
9     }  
10  },  
11  
12  
13   getById: async (req, res, next) => {  
14     try {  
15       res.json(await facultyService.getById(req.params.id));  
16     } catch (error) {  
17       next(error)  
18     }  
19  },  
20  
21   addFaculty: async (req, res, next) => {  
22     try {  
23       const newFaculty = req.body;  
24       res.json(await facultyService.addFaculty(newFaculty));  
25     } catch (error) {  
26       next(error)  
27     }  
28  }  
29 }  
30
```


Routers

```
routes > JS facultyRouter.js > ...
1  const express = require('express');
2  const facultyController = require('../controllers/facultyController');
3  const errors = require('../helpers/errors');
4
5  module.exports = () => {
6    let router = express.Router();
7
8    router.route('/')
9      .get(facultyController.getAll)
10     .post(facultyController.addFaculty)
11     .all((req, res, next) => res.error(errors.methodNotAllowed));
12
13    router.route('/:id')
14      .get(facultyController.getById)
15      .all((req, res, next) => res.error(errors.methodNotAllowed));
16
17    return router;
18  };
```


app.js

```
JS app.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser');
3
4  const errors = require('./helpers/errors');
5  const errorController = require('./global-controllers/errorController');
6  const facultyRoutes = require('./routers/facultyRouter')();
7
8  let app = express();
9
10 app.use(bodyParser.json({ extended: false }));
11 app.use('/faculties/', facultyRoutes);
12 app.use((req, res, next) => {
13   res.error(errors.resourceNotFound)
14 })
15 app.use(errorController);
16
17 app.listen(3000, () => console.log(`Server running at port 3000`));
18
```

Демонстрация работы (404, 405)

GET ⌵ http://localhost:3000/qwerty

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Body Cookies Headers (7) Test Results 🌐 Status: 404 Not Found

Pretty Raw Preview Visualize JSON ⌵ ⇌

```
1 {  
2   "message": "Resource not found",  
3   "code": "resource_not_found",  
4   "status": 404  
5 }
```

PUT ⌵ http://localhost:3000/faculties/MC

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body Cookies Headers (7) Test Results 🌐 Status: 405 Method Not Allowed

Pretty Raw Preview Visualize JSON ⌵ ⇌

```
1 {  
2   "message": "Method not allowed",  
3   "code": "method_not_allowed",  
4   "status": 405  
5 }
```

Демонстрация работы

GET http://localhost:3000/faculties/

Params Authorization Headers (7) Body Pre-request Script


Body Cookies Headers (7) Test Results


Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "faculty": "NEW",
4     "faculty_name": "NEW"
5   },
6   {
7     "faculty": "ИДиП",
8     "faculty_name": "Издательское дело и по
9   },
10  {
11    "faculty": "ИТ",
12    "faculty_name": "Информационных технол
13  },
```

GET http://localhost:3000/faculties/MC

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Body Cookies Headers (7) Test Results  Status: 404 Not Found

Pretty Raw Preview Visualize JSON 

```
1 {
2   "message": "Entity not found",
3   "code": "entity_not_found",
4   "status": 404
5 }
```

Демонстрация работы

POST ▼ http://localhost:3000/faculties

Params Authorization Headers (9) **Body** ● Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {
2   "faculty": "MC",
3   "faculty_name": "Машиностроения"
4 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▼ ↔

```
1 {
2   "faculty": "MC",
3   "faculty_name": "Машиностроения"
4 }
```

POST ▼ http://localhost:3000/faculties

Params Authorization Headers (9) **Body** ● Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {
2   "faculty": "MC",
3   "faculty_name": "Машиностроения"
4 }
```

Body Cookies Headers (7) Test Results 🌐 Status: 400 Bad Request

Pretty Raw Preview Visualize JSON ▼ ↔

```
1 {
2   "message": "Already exists",
3   "code": "already_exists",
4   "status": 400
5 }
```

Демонстрация работы

GET ⌵ http://localhost:3000/faculties/MC

Params Authorization Headers (7) Body Pre-request Script Tests Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies Headers (7) Test Results 🌐 Status: 200 OK

Pretty Raw Preview Visualize JSON ⌵ ≡

```
1 {
2   "faculty": "MC",
3   "faculty_name": "Машиностроения"
4 }
```