



Web-технологии

Применение Express, NestJS, RESTful, Ajax

Содержание

- Использование Express, Кoa
- Использование шаблонов на основе PUG и EJS
- Cookies на сервере
- Реализация сессии
- Фреймворк NestJS
 - архитектура приложения
 - базовый пример на JS
 - основные возможности
 - подключение шаблонов
- RESTful на Node.js
- Ajax
- XMLHttpRequest, fetch
- CORS

<https://www.tutorialspoint.com/expressjs/>

<https://www.w3schools.com/>

<http://xmlhttprequest.ru/>

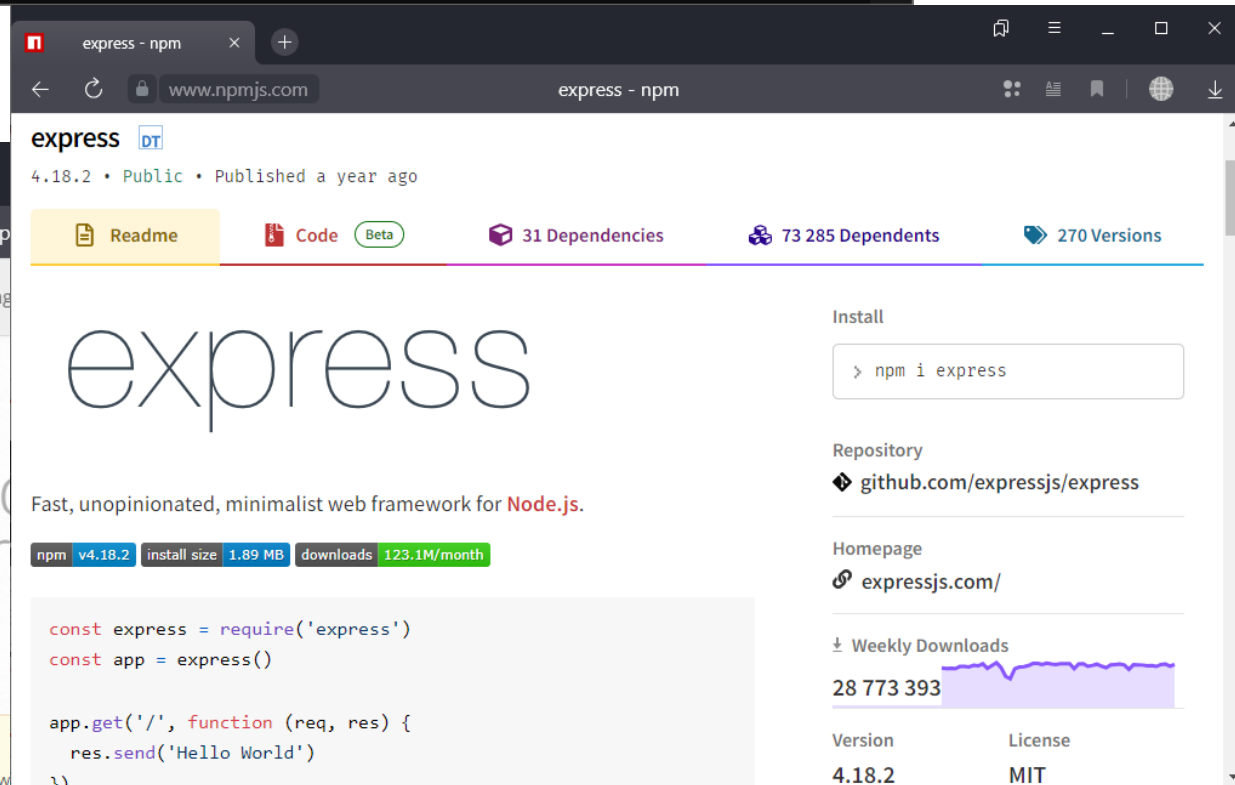
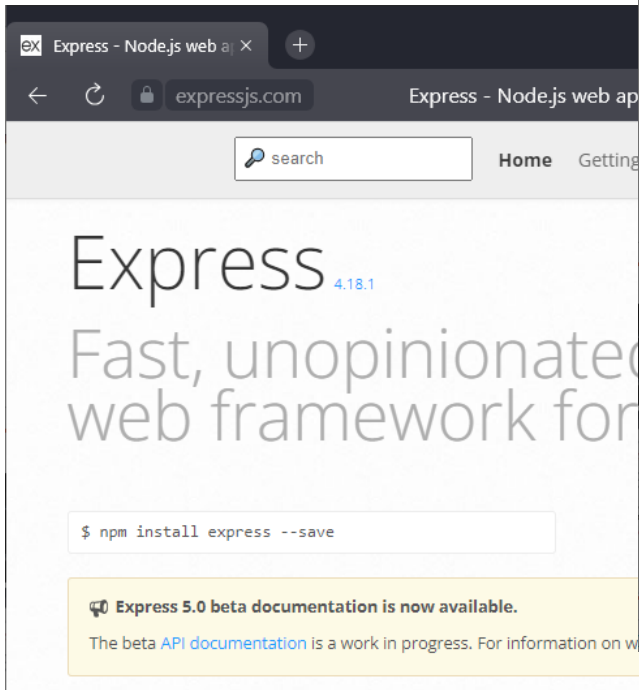
npm i express

3

```
Command Prompt
C:\Users\serge\WebstormProjects\test>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN test@1.0.0 No description
npm WARN test@1.0.0 No repository field.

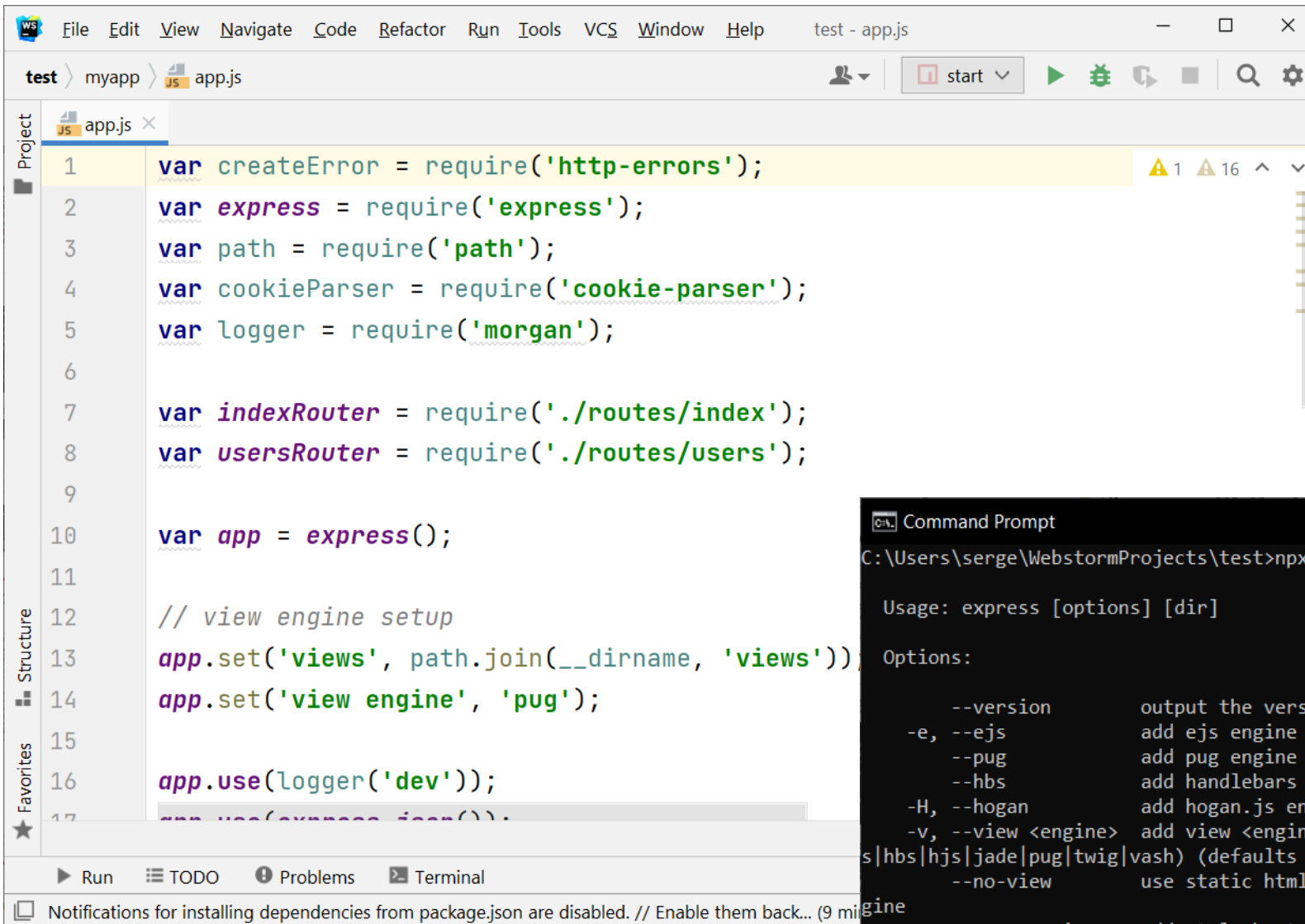
+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 2.657s
found 0 vulnerabilities
```

<https://expressjs.com/>



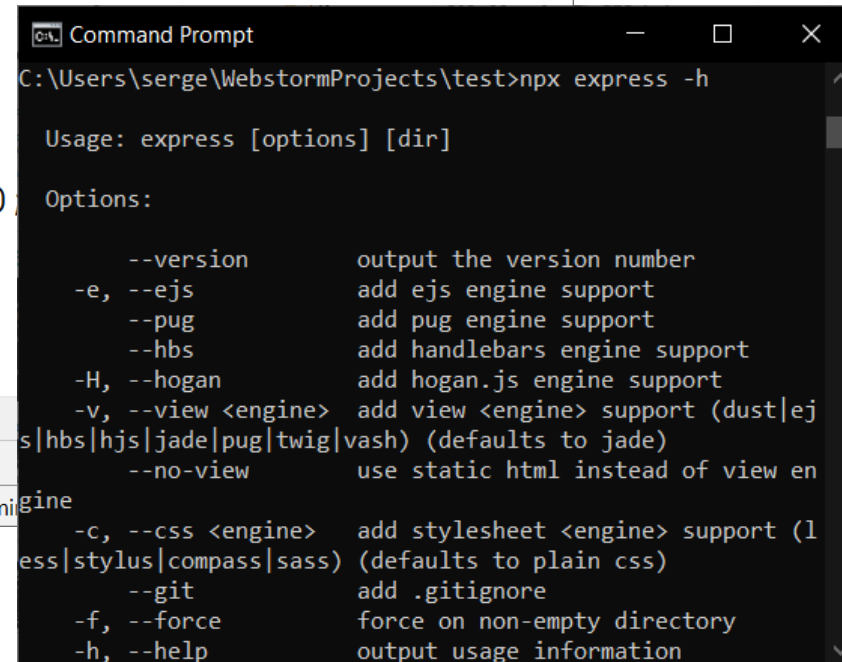
Генератор приложений с express

4



The screenshot shows an IDE window titled 'test - app.js'. The file 'app.js' is open, displaying the following code:

```
1 var createError = require('http-errors');
2 var express = require('express');
3 var path = require('path');
4 var cookieParser = require('cookie-parser');
5 var logger = require('morgan');
6
7 var indexRouter = require('./routes/index');
8 var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view engine', 'pug');
15
16 app.use(logger('dev'));
17 app.use(express.json());
18 app.use(express.urlencoded({ extended: false }));
19 app.use(cookieParser());
20 app.use(indexRouter);
21 app.use(usersRouter);
22
23 app.listen(3000, function() {
24   console.log('Example app listening on port 3000');
25 });
```



The screenshot shows a Command Prompt window with the command `npx express -h` executed. The output displays the usage and options for the `express` command:

```
C:\Users\serge\WebstormProjects\test>npx express -h

Usage: express [options] [dir]

Options:

  --version           output the version number
  -e, --ejs           add ejs engine support
  --pug              add pug engine support
  --hbs              add handlebars engine support
  -H, --hogan        add hogan.js engine support
  -v, --view <engine> add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
  --no-view          use static html instead of view engine
  -c, --css <engine> add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
  --git              add .gitignore
  -f, --force        force on non-empty directory
  -h, --help         output usage information
```

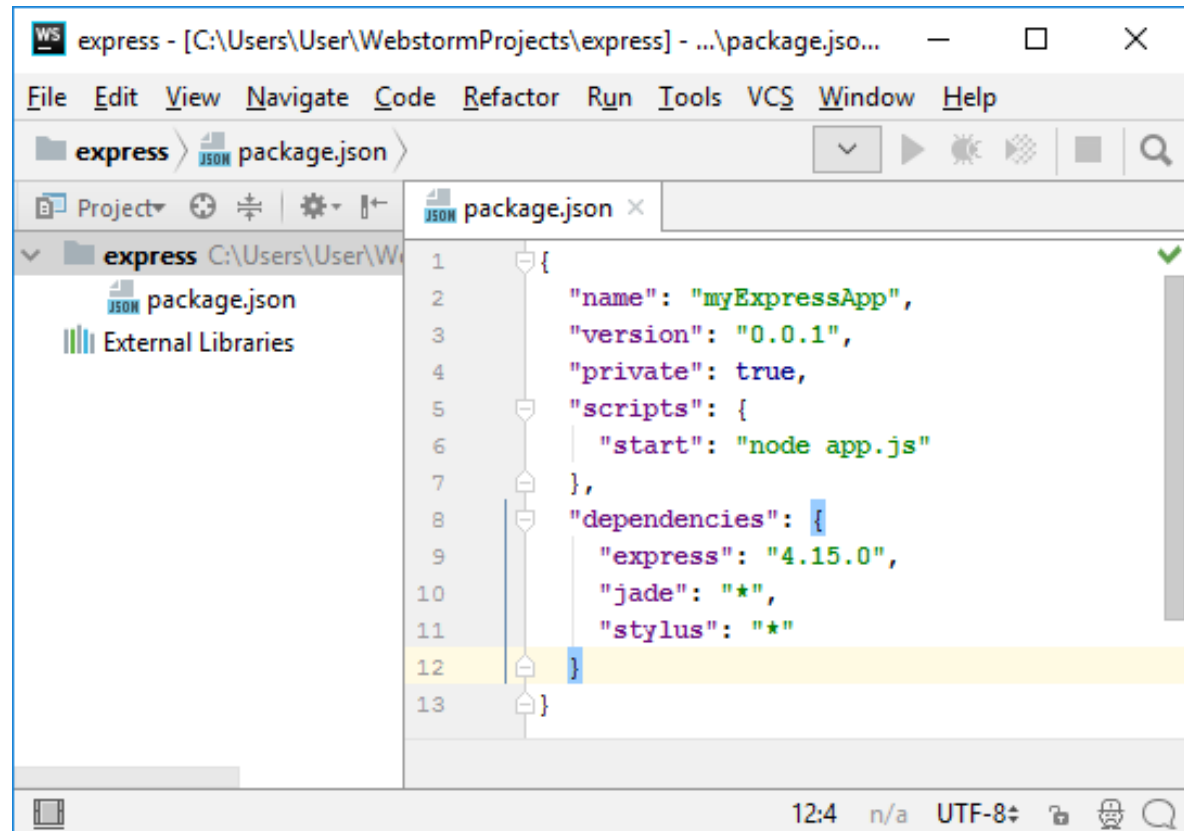
`npm install express-generator`
`npx express --view=pug myapp`

package.json

5

```
{  
  "name": "myExpressApp",  
  "version": "0.0.1",  
  "private": true,  
  "scripts": {  
    "start": "node app.js"  
  },  
  "dependencies": {  
    "express": "*",  
    "pug": "*"   
  }  
}
```

В предыдущей редакции был **jade** – он устарел и запрещён к использованию. **Заменить на pug!**

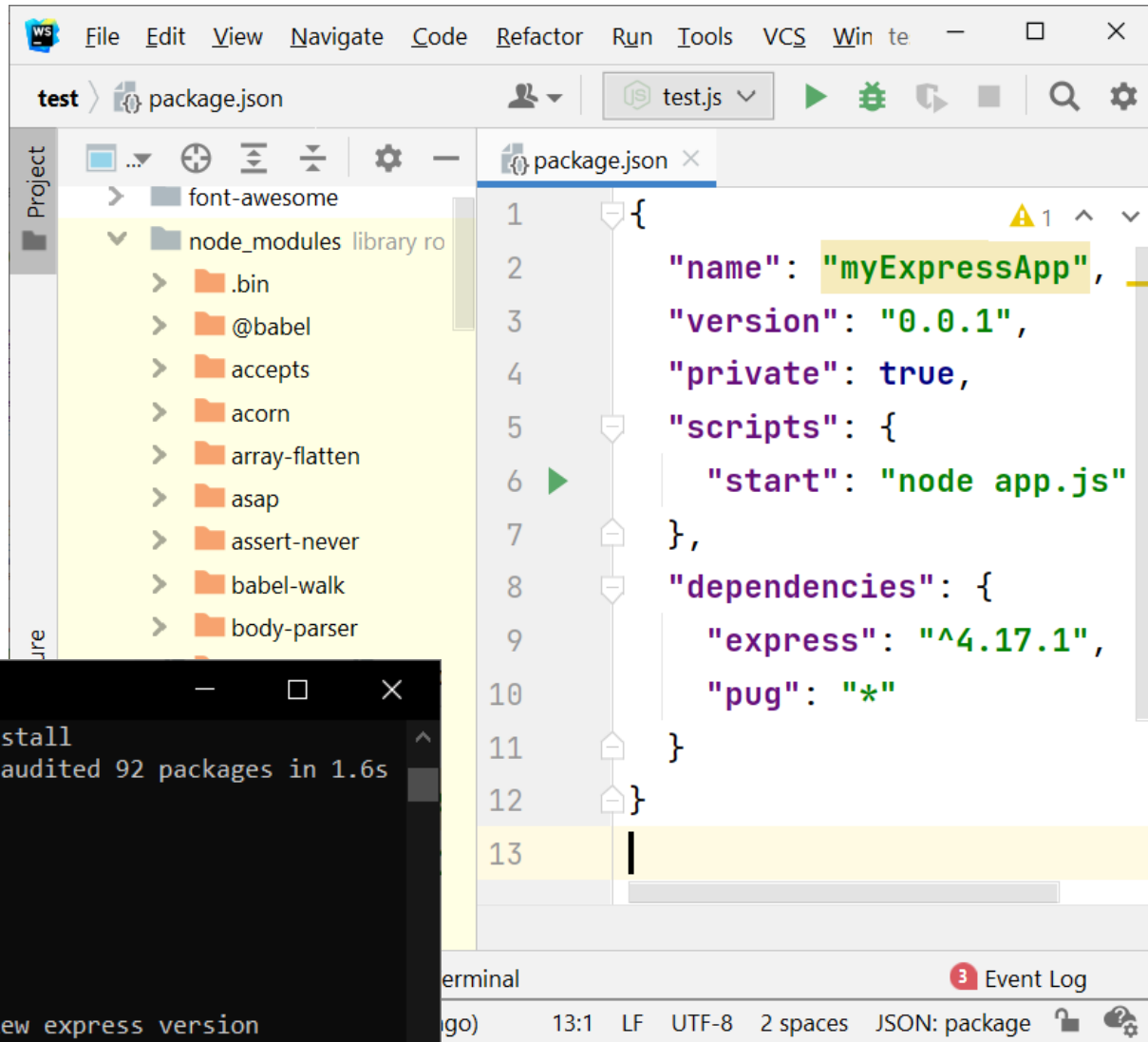


npm install

6

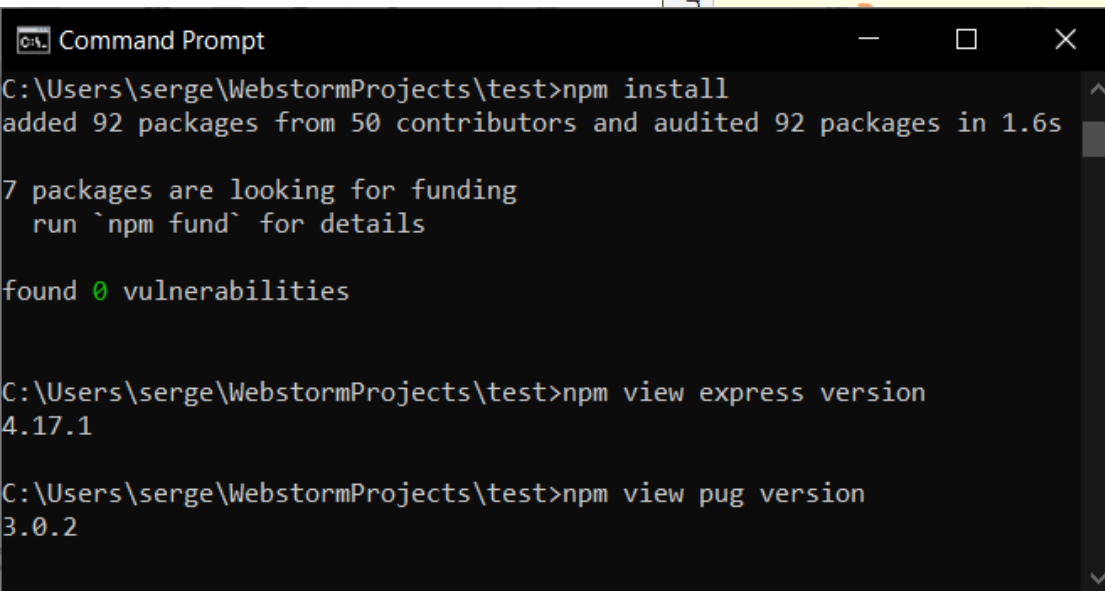
Установка пакетов,
описанных в
package.json

Комментарий
node_modules не
загружаются в
репозитории
исходных кодов!



The screenshot shows an IDE window with a project explorer on the left and a code editor on the right. The project explorer shows a folder structure with 'font-awesome' and 'node_modules' (which contains subfolders like '.bin', '@babel', 'accepts', 'acorn', 'array-flatten', 'asap', 'assert-never', 'babel-walk', and 'body-parser'). The code editor displays the contents of 'package.json':

```
1 {  
2   "name": "myExpressApp",  
3   "version": "0.0.1",  
4   "private": true,  
5   "scripts": {  
6     "start": "node app.js"  
7   },  
8   "dependencies": {  
9     "express": "^4.17.1",  
10    "pug": "*"br/>11  }  
12 }  
13
```



The screenshot shows a Command Prompt window with the following output:

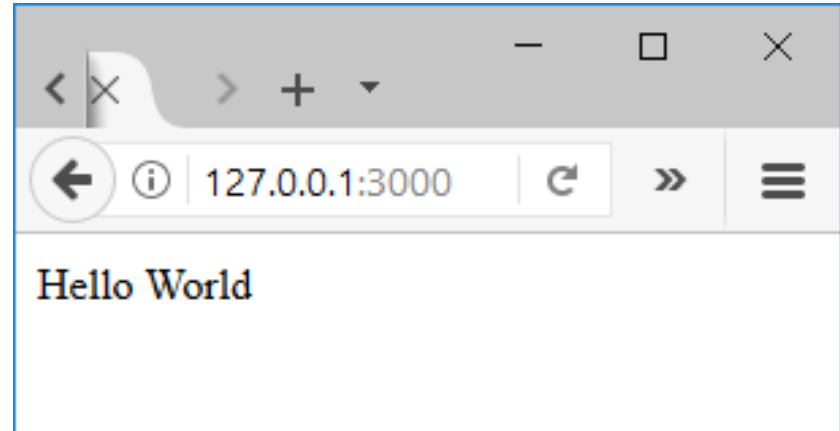
```
C:\Users\serge\WebstormProjects\test>npm install  
added 92 packages from 50 contributors and audited 92 packages in 1.6s  
  
7 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
  
C:\Users\serge\WebstormProjects\test>npm view express version  
4.17.1  
  
C:\Users\serge\WebstormProjects\test>npm view pug version  
3.0.2
```

Простейшее приложение express

7

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

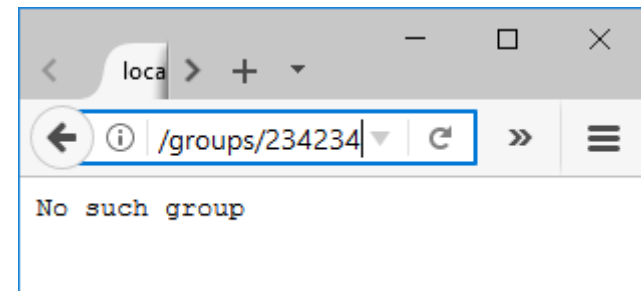
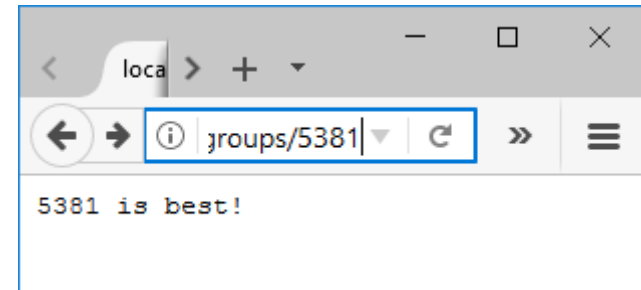
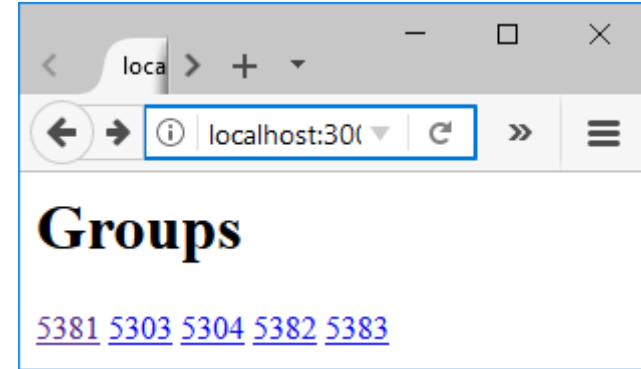
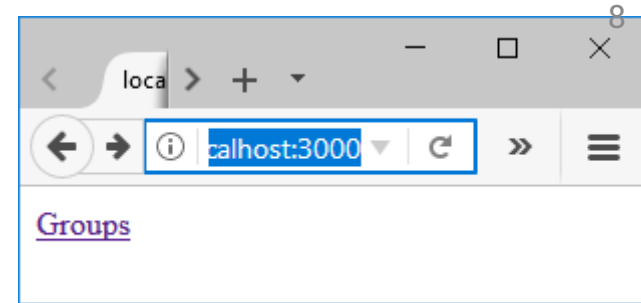
```
app.get('/', (req, res) => {  
  res.send('Hello World')  
}).listen(3000);
```



Усложняем

```
let express = require("express");
let server = express();
let port = process.env.PORT || 3000;
```

```
let groups = ["5381", "5303", "5304", "5382", "5383"];
server.get("/groups/:num", (req, res, next) => {
  let number = req.params.num;
  for (value of groups)
    if (value === number) {
      res.end(`${value} is best!`);
      return;
    }
  next();
});
server.get("/groups/", (req, res) => {
  let body = "<h1>Groups</h1>";
  for (value of groups)
    body += `<a href="/groups/${value}">${value}</a>`;
  res.end(body);
});
server.get("/groups/*", (req, res) => {
  res.end("No such group");
});
server.get("/", (req, res) => {
  res.end("<a href='/groups/'>Groups</a>");
});
server.listen(port);
```



Делим на модули (1)

9

app.js

```
let express = require("express");
let server = express();
let routes = require("./routes");

let groups = ["5381", "5303", "5304", "5382", "5383"];
server.get("/groups/:num", (req, res, next) => {
  let number = req.params.num;
  for (value of groups)
    if (value === number) {
      res.end(`${value} is best!`);
      return // Иначе будет ошибка
    }
  next();
});
server.get("/groups/", (req, res) => {
  let body = "<h1>Groups</h1>";
  for (value of groups)
    body += `<a href="/groups/${value}">${value}</a>`;
  res.end(body);
});
server.use("/", routes);
server.listen(3000);
```

routes.js

```
let express = require("express");
let router = express.Router();

router.get("/groups/*", (req, res) => {
  res.end("No such group");
});
router.get("/", (req, res) => {
  res.end("<a href='/groups/'>Groups</a>");
});
module.exports = router;
```

Делим на модули (2)

10

app.js

```
let express = require("express");
let server = express();
let routes = require("./routes");

server.use("/", routes);
server.listen(3000);
```

Здесь **end** заменён на **send**!

routes.js

```
let express = require("express");
let router = express.Router();
let groups = ["5381", "5303", "5304", "5382", "5383"];

router.get("/groups/:num", (req, res, next) => {
  let number = req.params.num;
  for (value of groups)
    if(value === number) {
      res.send(`${value} is best!`);
      return // Иначе будет ошибка
    }
  next();
});

router.get("/groups/", (req, res)=>{
  let body = "<h1>Groups</h1>";
  for (value of groups)
    body += `<a href="/groups/${value}">${value}</a>`;
  res.send(body);
});

router.get("/groups/*", (req, res)=>{
  res.send("No such group");
});

router.get("/", (req, res)=>{
  res.send("<a href='/groups/'>Groups</a>");
});
module.exports = router;
```

Методы response в express

11

- **append** – Добавляет указанное значение в поле заголовка HTTP-ответа
- **attachment** – Устанавливает в поле заголовка содержимого HTTP-ответа значение “вложение”
- **cookie** – Задает значение cookie
- **clearCookie** – Удаляет cookie
- **download** – Передает файл в виде “вложения”
- **end** – Завершает процесс ответа
- **format** – Выполняет согласование содержимого с заголовком HTTP Ассерт для объекта запроса, если он присутствует
- **get** – Возвращает заголовок HTTP-ответа
- **json** – Отправляет ответ в формате JSON
- **links** – Соединяет ссылки
- **location** – Устанавливает HTTP-заголовок местоположения ответа в указанный параметр пути
- **redirect** – Перенаправляет на URL
- **render** – Отрисовывает представление и отправляет отрисованную HTML-строку клиенту
- **send** – Отправляет HTTP-ответ
- **sendFile, sendStatus, set, status, type, vary, jsonp**

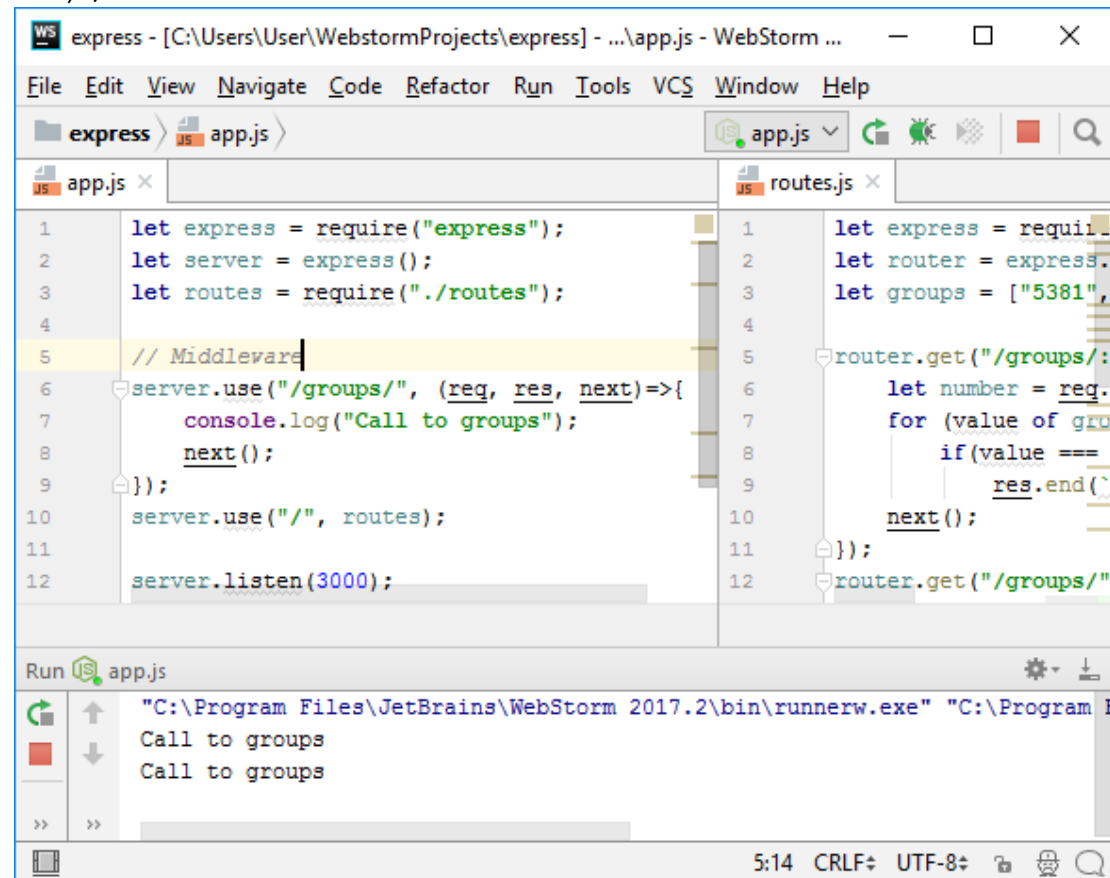
Middleware (1)

12

```
let express = require("express");
let server = express();
let routes = require("./routes");

// Middleware
server.use("/groups/", (req, res, next)=>{
    console.log("Call to groups");
    next();
});
server.use("/", routes);

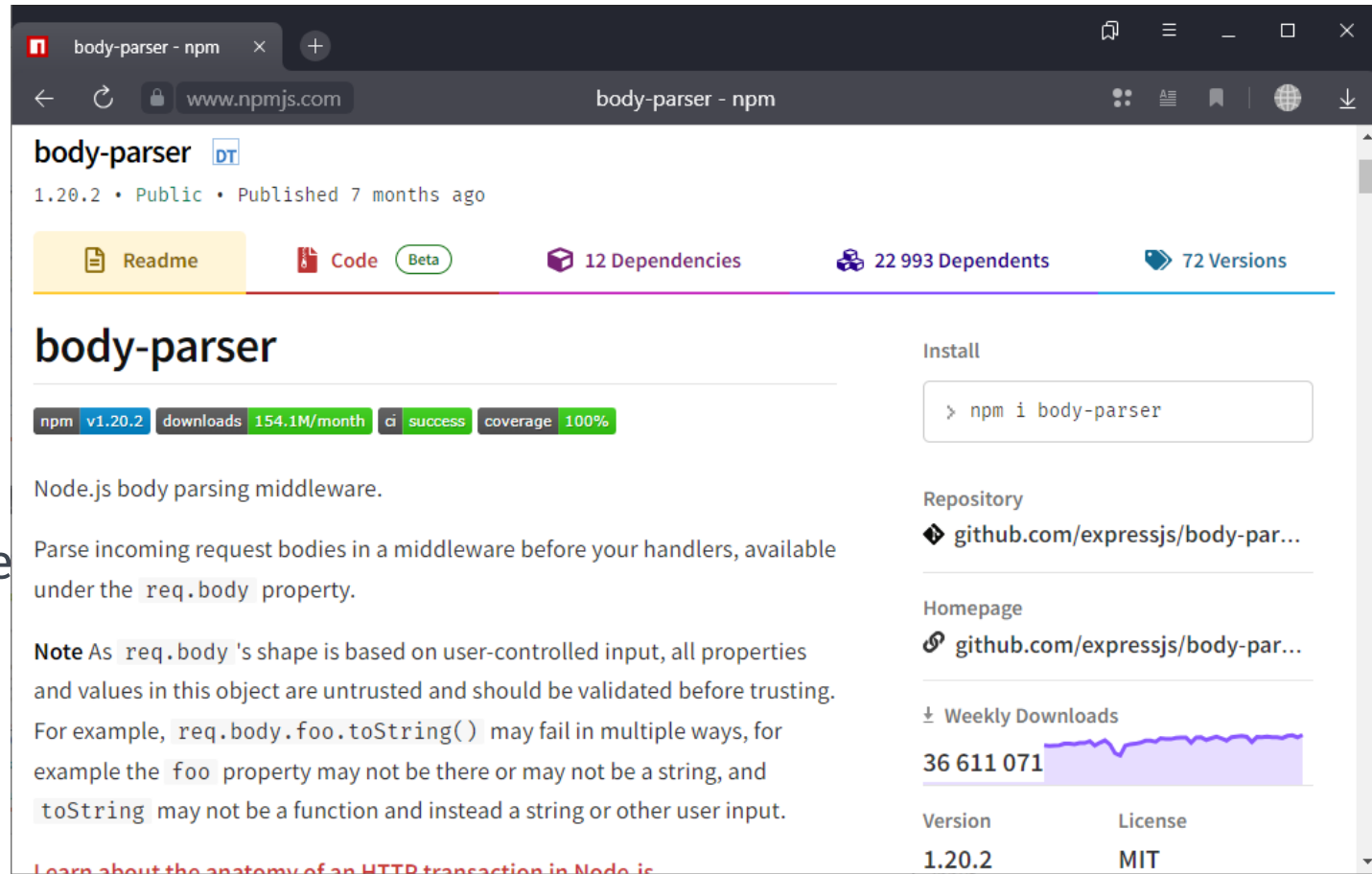
server.listen(3000);
```



Middleware (2)

13

- body-parser
- compression
- connect-rid
- cookie-parser
- cookie-session
- cors
- csurf
- errorhandler
- method-override
- morgan
- multer
- response-time
- serve-favicon
- serve-index
- serve-static
- session
- timeout
- vhost



The screenshot shows the npm page for the `body-parser` package. The page header includes the package name, version (1.20.2), and publication status (Public, 7 months ago). Below this, there are links to the README, Code, and a Beta badge, along with statistics: 12 Dependencies, 22,993 Dependents, and 72 Versions. The main section features the package name, a description, and a list of badges for npm version, downloads (154.1M/month), CI success, and coverage (100%). The description states that `body-parser` is Node.js body parsing middleware used to parse incoming request bodies. A note warns that the `req.body` object is untrusted and should be validated. The right sidebar contains an 'Install' section with a command to install the package, a 'Repository' link to the GitHub repository, a 'Homepage' link, and a 'Weekly Downloads' chart showing a steady increase in downloads over time. A table at the bottom lists the current version (1.20.2) and the license (MIT).

body-parser `DT`

1.20.2 • Public • Published 7 months ago

Readme Code Beta 12 Dependencies 22 993 Dependents 72 Versions

body-parser

npm v1.20.2 downloads 154.1M/month ci success coverage 100%

Node.js body parsing middleware.

Parse incoming request bodies in a middleware before your handlers, available under the `req.body` property.

Note As `req.body`'s shape is based on user-controlled input, all properties and values in this object are untrusted and should be validated before trusting. For example, `req.body.foo.toString()` may fail in multiple ways, for example the `foo` property may not be there or may not be a string, and `toString` may not be a function and instead a string or other user input.

Learn about the anatomy of an HTTP transaction in Node.js

Install

```
> npm i body-parser
```

Repository

github.com/expressjs/body-par...

Homepage

github.com/expressjs/body-par...

Weekly Downloads

36 611 071

Version	License
1.20.2	MIT

```
let express = require('express')
let app = express()
let cookieParser = require('cookie-parser')
app.use(cookieParser())
```

<https://expressjs.com/en/resources/middleware/cookie-parser.html>

Обработка ошибок (1)

```
let app = express();  
let express = require("express");  
app.use(function(err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send('Something broke!');  
});
```

Обработка ошибок (2)

```
let app = express();
let express = require("express");
let bodyParser = require('body-parser');
let methodOverride = require('method-override');

app.use(bodyParser());
app.use(methodOverride());
app.use(logErrors);
app.use(clientErrorHandler);
app.use(errorHandler);
/* запись информации о запросах и ошибках в stderr */
function logErrors(err, req, res, next) {
  console.error(err.stack);
  next(err);
}
/* передача ошибки клиенту */
function clientErrorHandler(err, req, res, next) {
  if (req.xhr) {
    res.status(500).send({ error: 'Something failed!' });
  } else {
    next(err);
  }
}
/* обобщающая функция ошибки */
function errorHandler(err, req, res, next) {
  res.status(500);
  res.render('error', { error: err });
}
```

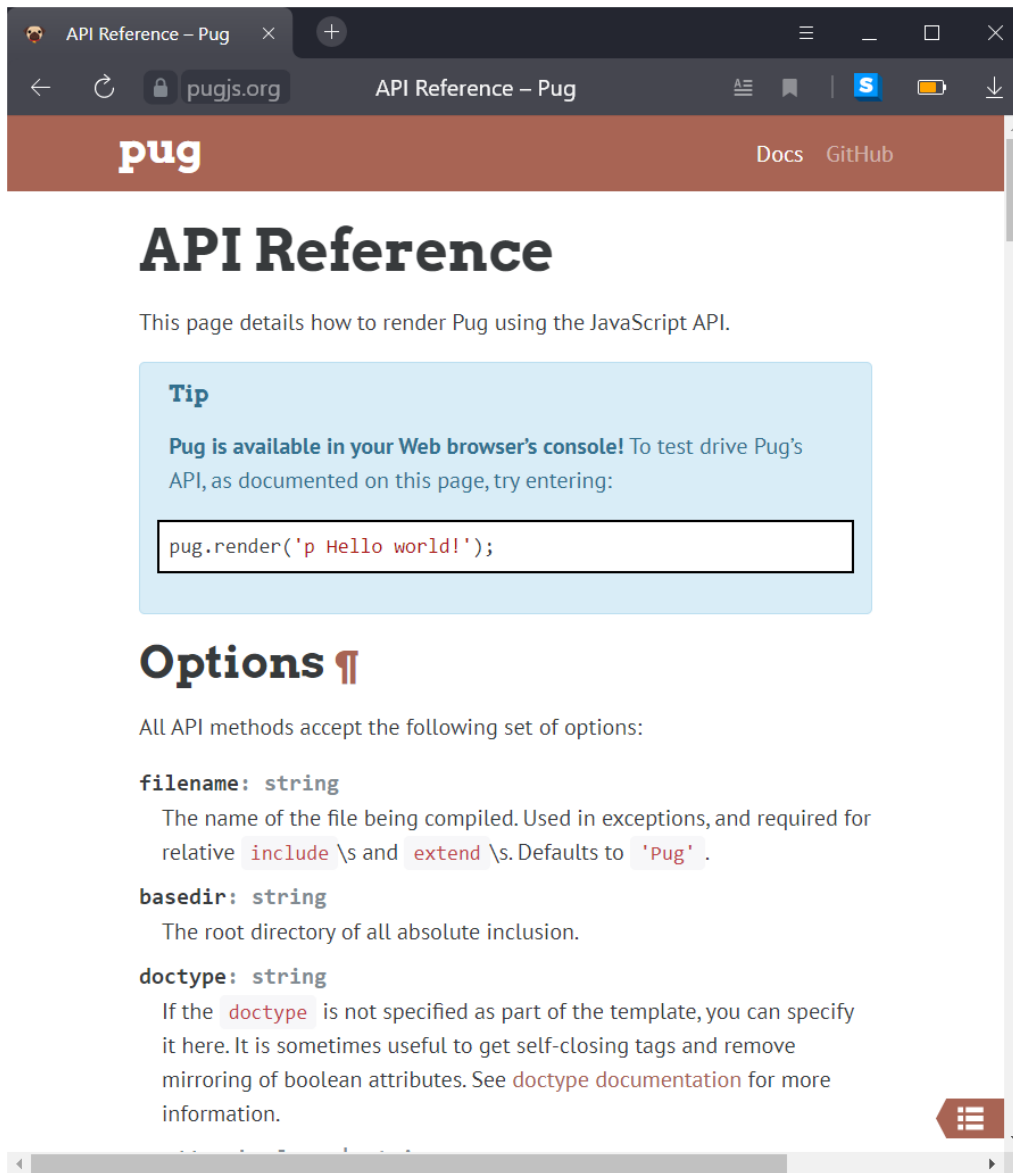
Предусмотрен
встроенный в
express
обработчик
ошибок,
вызываемый по
команде **next(err)**

Шаблоны HTML с использованием PUG

<https://pugjs.org/api/reference.html>

- Работает на клиентской стороне
- Отличная читабельность
- Гибкая система отступов
- Раскрытие блоков
- Примеси
- Статические инклюды
- Интерполяция в атрибутах
- Контекстные сообщения об ошибках
- Кеширование в памяти
- Комбинирование динамических и статических CSS-классов
- Поддержка Express JS «из коробки»
- Блочные комментарии

npm install --save pug



The screenshot shows a web browser window with the address bar displaying `pugjs.org`. The page title is "API Reference – Pug". The browser's address bar also shows "API Reference – Pug". The page content includes a header with the "pug" logo and links to "Docs" and "GitHub". The main heading is "API Reference". Below this, a paragraph states: "This page details how to render Pug using the JavaScript API." A "Tip" box contains the text: "Pug is available in your Web browser's console! To test drive Pug's API, as documented on this page, try entering:" followed by a code input field containing `pug.render('p Hello world!');`. Below the tip box, the heading "Options ¶" is followed by a paragraph: "All API methods accept the following set of options:". The options listed are:

- filename:** string. The name of the file being compiled. Used in exceptions, and required for relative `include` \s and `extend` \s. Defaults to `'Pug'`.
- basedir:** string. The root directory of all absolute inclusion.
- doctype:** string. If the `doctype` is not specified as part of the template, you can specify it here. It is sometimes useful to get self-closing tags and remove mirroring of boolean attributes. See [doctype documentation](#) for more information.

Использование PUG (1)

17

groups.pug

```
doctype html
html
  head
    meta(charset="utf-8")
    title= name
  body
    h1= name
    ul
      - for(var key in groups)
        - value=groups[key]
        li
          a(href=value)=value
```

groups.js

```
let groups = ["5381", "5303", "5304", "5382", "5383"];
```

app.js

```
let express = require("express");
let server = express();
let routes = require("./routes");
let groups = require("./groups");
server.set("view engine", "pug");
server.set("views", `./views`);

server.get("/groups/", (req, res) => {
  res.render("groups", {
    name: "Группы",
    groups: groups
  });
});
server.use("/", routes);

server.listen(3000);
```

Перенесен
из router.js

Использование PUG (2)

18

Группы

- [5381](#)
- [5303](#)
- [5304](#)
- [5382](#)
- [5383](#)

```
3      head
4        meta(charset="utf-8")
5        title= name
6      body
7        h1= name
8        ul
9          - for(var key in groups)
10            - value=groups[key]
11            li
12              a(href=value)=value
```

```
1 let groups = ["5381", "5303", "5304", "5382", "5383"];
2
3 module.exports = groups;
```

groups

objects\express] - ...\app.js - WebStorm 2017.2

actor Run Tools VCS Window Help

app.js

routes.js

app.js

```
1 let express = require("express");
2 let server = express();
3 let routes = require("./routes");
4 let groups = require("./groups");
5 server.set("view engine", "jade");
6 server.set("views", `./views`);
7
8 server.get("/groups/", (req, res) => {
9   res.render("groups", {
10     name: "Группы",
11     groups: groups
12   });
13 });
14 server.use("/", routes);
15
16 server.listen(3000);
```

callback for get()

Использование PUG (3)

19

groups.pug

```
doctype html
html
  head
    link(href='/public/w3.css', rel='stylesheet')
    meta(charset="utf-8")
    title= name
  body
    h1= name
    ul
```

app.js

```
let express = require("express");
let server = express();
let routes = require("./routes");
let groups = require("./groups");
server.use('/public', express.static('public'));
server.set("view engine", "pug");
server.set("views", `./views`);

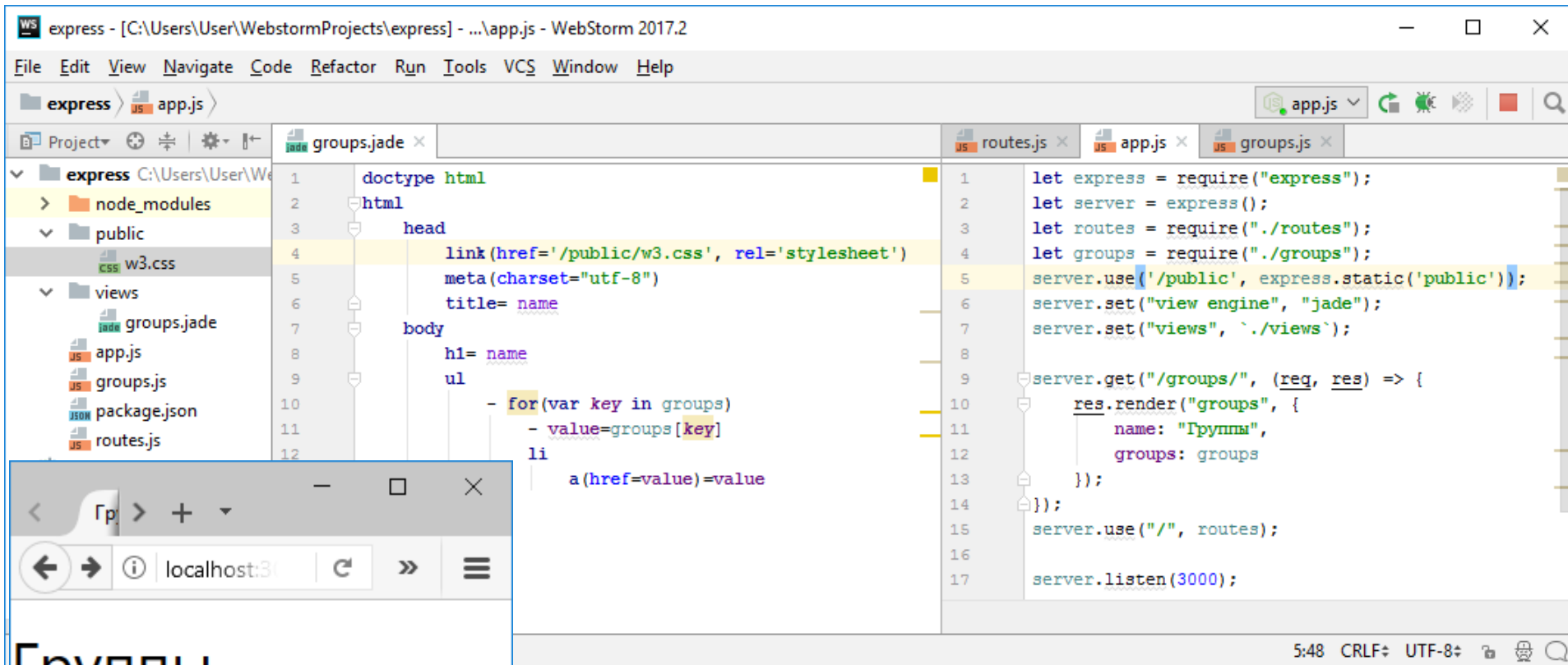
server.get("/groups/", (req, res) => {
  res.render("groups", {
    name: "Группы",
    groups: groups
  });
});

server.use("/", routes);

server.listen(3000);
```

Использование PUG (4)

20



Допустимо использование нескольких статических папок

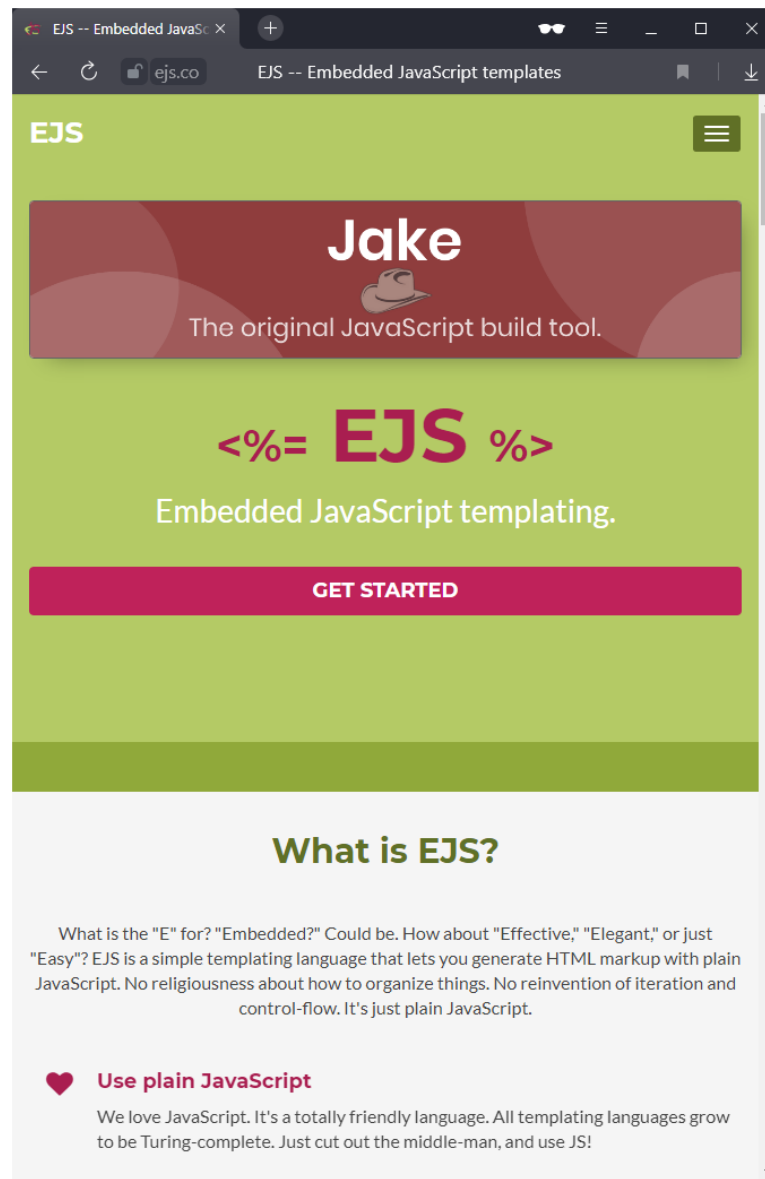
- [5381](#)
- [5303](#)
- [5304](#)
- [5382](#)
- [5383](#)

Шаблоны HTML с использованием EJS

<https://ejs.co/>

- Быстрая компиляция и рендеринг
- Простые теги шаблонов: `<% %>`
- Пользовательские разделители
 - например, используйте `[? ?]` вместо `<% %>`
- Поддержка подшаблонов
- Поддержка CLI
- Поддержка как JS сервера, так и браузера
- Статическое кэширование промежуточного JavaScript
- Статическое кэширование шаблонов
- Компилируется с Express

`npm install --save ejs`



- Формирование данных по шаблону

```
let ejs = require('ejs');  
let people = ['geddy', 'neil', 'alex'];  
let html = ejs.render('<b><%= people.join(", "); %></b>', {people: people});  
html // <b>geddy, neil, alex</b>
```

- Командная строка

- `ejs ./template_file.ejs -f data_file.json -o ./output.html`

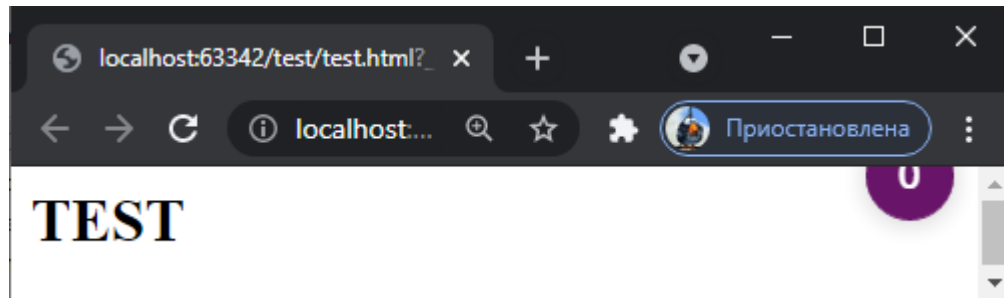
- В браузере

```
<script src="ejs.js"></script>  
<script>  
  let people = ['geddy', 'neil', 'alex'];  
  let html = ejs.render('<b><%= people.join(", "); %></b>', {people: people});  
</script>
```

Варианты применения EJS Браузер

compile

```
let str = `  
<% if (user) { %>  
  <h2><%= user.name %></h2>  
<% } %>  
`
```



```
let template = ejs.compile(str);  
document.writeln(template({ user: { name: "TEST" }}));
```

render

```
let str = `  
<% if (user) { %>  
  <h2><%= user.name %></h2>  
<% } %>  
`
```

```
document.writeln(ejs.render(str, { user: { name: "TEST" }}));
```

Поддерживается
параметр **options**

Варианты применения EJS Сервер


test.ejs

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

test.js

```
let ejs = require('ejs');
ejs.renderFile("test.ejs", { user: { name: "TEST" } }, function(err, str) {
  console.log(str) // <h2>TEST</h2>
});
```

Поддерживается
параметр **options**



ejs.renderFile не работает в браузере из-за отсутствия доступа к **fs**.

- `<%`
 - Тег «скриплета» - контроль потока, нет output
- `<%=`
 - Тег «whitespace slurping» удаляет лишние пробелы «до»
- `<%=`
 - Выводит результаты в шаблон (HTML escaped)
- `<%-`
 - Выводит результаты в шаблон (unescaped)
- `<%=`
 - Комментарий, не выполняется, нет output
- `<%%`
 - Выводит литерал '`<%`'
- `%>`
 - Закрывающий тег
- `-%>`
 - Тег «newline slurp» обрезает пустые строки в конце
- `_%>`
 - Тег «whitespace slurping» удаляет лишние пробелы «после»

EJS. Примеры forEach, include

26

Пример 1

```
<ul>  
  <% users.forEach(function(user){ %>  
    <%- include('user/show', {user: user}); %>  
  <% }); %>  
</ul>
```

Пример 2

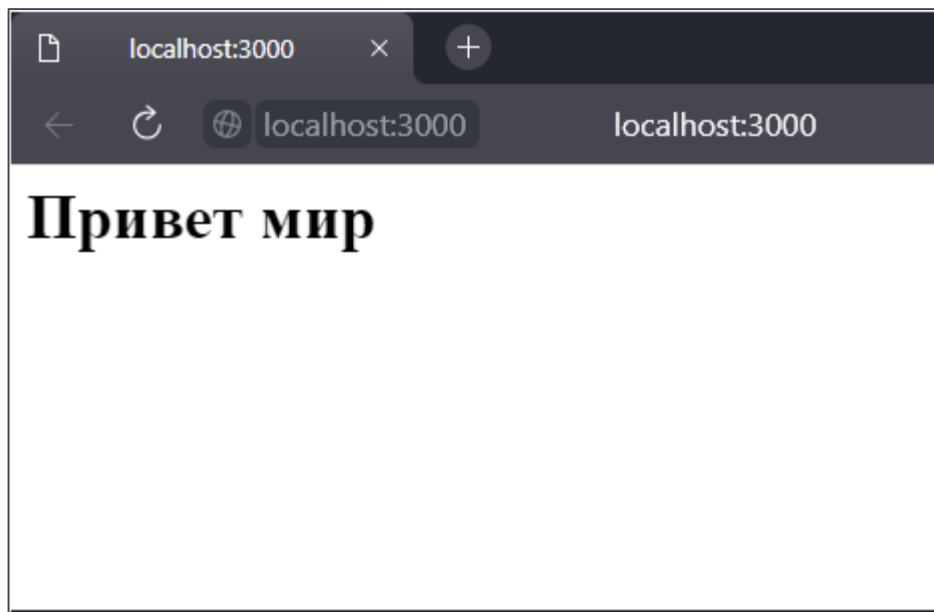
```
<%- include('header'); -%>  
<h1>  
  Title  
</h1>  
<p>  
  My page  
</p>  
<%- include('footer'); -%>
```

Использование EJS в Express

```
const express = require("express");
const app = express();
app.set("view engine", "ejs");
app.use("/", function(request, response){
  response.render("test", {
    title: "Привет мир"
  });
});
app.listen(3000);
```

views/test.ejs

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <h1><%=title%></h1>
</body>
</html>
```



Неявная настройка папки **views**

Использование body-parser и PUG (1) 28

form.pug

```
html
  head
    meta(charset="utf-8")
    link(href='/public/w3.css', rel='stylesheet')
    title Проверка body-parser
  body
    form(action = "/", method = "POST")
      div
        label(for = "text") Текст
        br
        input(name = "text" value = "Текст")
      br
      div
        label(for = "to") Адресат
        br
        input(name = "to" value = "Поль:")
      br
      button(type = "submit") Отправить
```

server.js

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use('/public', express.static('public'));
app.get('/', function(req, res){
  res.render('form');
});
app.set('view engine', 'pug');
app.set('views', './views');
// разбор application/json
app.use(bodyParser.json());
// разбор application/x-www-
app.use(bodyParser.urlencoded({ extended: true }));
app.post('/', function(req, res){
  console.log(req.body);
  res.send("Recieved your request!");
});
app.listen(3000);
```

Использование body-parser и PUG (2) ²⁹

The screenshot displays a web application running in a browser at `localhost:3000`. The browser shows a form with two text input fields labeled "Текст" and "Текст 2", an input field labeled "Адресат" with the value "Пользователь 2", and a "Отправить" (Submit) button. A message "Recieved your request!" is displayed below the form.

The background shows the WebStorm IDE with the following code:

form.pug

```
1  html
2    head
3      meta(charset="utf-8")
4      link(href="/public/w3.css", rel="stylesheet")
5      title Проверка body-parser
6    body
7      form(action = "/", method = "POST")
8        div
9          label(for = "text") Текст
10         br
11         input(name = "text" value = "Текст")
12       br
13       div
14         label(for = "to") Адресат
15         br
16         input(name = "to" value = "Пользователь")
17       br
18       button(type = "submit") Отправить
```

server.js

```
1  var express = require('express');
2  var bodyParser = require('body-parser');
3  var app = express();
4  app.use('/public', express.static('public'));
5  app.get('/', function(req, res){
6    res.render('form');
7  });
8  app.set('view engine', 'pug');
9  app.set('views', './views');
10 // парсер application/json
11 app.use(bodyParser.json());
12 // парсер application/x-www-
13 app.use(bodyParser.urlencoded({ extended: true }));
14 app.post('/', function(req, res){
15   console.log(req.body);
16   res.send("Recieved your request!");
17 });
18 app.listen(3000, function(){
19   console.log("Server is running on port 3000");
20 });
```

Run console output:

```
{ text: 'Текст', to: 'Пользователь' }
{ text: 'Текст 2', to: 'Пользователь 2' }
```

<https://expressjs.com/en/resources/middleware/body-parser.html>

Cookies (1)

30

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

app.get('/', (req, res)=>{
  res.cookie('it\'s name', 'it\'s value');
  res.end('Cookie set');
});

app.listen(3000);
```

The screenshot shows a web browser at `localhost:3000/` displaying the text "Cookie set". Below the browser window, the Chrome DevTools "Хранилище" (Storage) panel is open, showing the "Куки" (Cookies) section. A table lists cookies for `http://localhost:3000`. The cookie `it's name` is selected, and its details are shown on the right.

Имя	Домен	Путь	Тип
<code>_gaas</code>	<code>localhost</code>	<code>/</code>	<code>I</code>
<code>_io_lv</code>	<code>localhost</code>	<code>/</code>	<code>M</code>
<code>_io_uid_test</code>	<code>localhost</code>	<code>/</code>	<code>M</code>
<code>_io</code>	<code>localhost</code>	<code>/</code>	<code>M</code>
<code>_floor_uid</code>	<code>localhost</code>	<code>/</code>	<code>T</code>
<code>_gat</code>	<code>localhost</code>	<code>/</code>	<code>Si</code>
<code>_ga</code>	<code>localhost</code>	<code>/</code>	<code>T</code>
<code>_gid</code>	<code>localhost</code>	<code>/</code>	<code>M</code>
<code>_io_conversion...</code>	<code>localhost</code>	<code>/</code>	<code>M</code>
<code>_io_s</code>	<code>localhost</code>	<code>/</code>	<code>Si</code>
<code>_io_un_localhost</code>	<code>localhost</code>	<code>/</code>	<code>M</code>
<code>_ym_isad</code>	<code>localhost</code>	<code>/</code>	<code>M</code>
<code>_ym_uid</code>	<code>localhost</code>	<code>/</code>	<code>Si</code>
<code>_ym_visorc_164...</code>	<code>localhost</code>	<code>/</code>	<code>Si</code>
<code>it's name</code>	<code>localhost</code>	<code>/</code>	<code>C</code>
<code>name</code>	<code>localhost</code>	<code>/</code>	<code>C</code>

Details for `it's name`:

- CreationTime: "Sun, 01 Oct 2017 14:14:58 GMT"
- Domain: "localhost"
- Expires: "Сессионная"
- HostOnly: **true**
- HttpOnly: **false**
- LastAccessed: "Sun, 01 Oct 2017 14:14:58 GMT"
- Path: `/`
- Secure: **false**

Cookies (2)

31

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

app.get('/', (req, res)=>{
  res.cookie('name1', 'value', {
    expire: 36000 + Date.now()
  });
  res.cookie('name2', 'value', {
    maxAge: 36000
  });
  res.end('Cookie set');
}).listen(3000);
```

Срок исчисляется в
миллисекундах

The screenshot shows a web browser at localhost:3000 displaying the text 'Cookie set'. The Chrome DevTools 'Cookies' panel is open, showing a list of cookies for the domain 'localhost'. The cookie 'name2' is selected, and its details are shown on the right.

Имя	Домен	Путь	Тип
_io_iv	localhost	/	M
_io_uid_test	localhost	/	M
_io	localhost	/	M
_floor_uid	localhost	/	T
_gat	localhost	/	S
_ga	localhost	/	T
_gid	localhost	/	M
_io_conversion...	localhost	/	M
_io_s	localhost	/	S
_io_un_localhost	localhost	/	M
_ym_isad	localhost	/	M
_ym_uid	localhost	/	S
_ym_visorc_164...	localhost	/	S
name1	localhost	/	C
name2	localhost	/	S
Webstorm-aeb...	localhost	/	S

Details for 'name2':

- name2: "value"
- CreationTime: "Sun, 01 Oct 2017 14:17:25 GMT"
- Domain: "localhost"
- Expires: "Sun, 01 Oct 2017 14:18:01 GMT"
- HostOnly: true
- HttpOnly: false
- LastAccessed: "Sun, 01 Oct 2017 14:17:25 GMT"
- Path: "/"
- Secure: false

Сессия (1)

html

msg.pug

32

head

meta (charset="utf-8")

title Сессия

body

h1 Счётчик

-if (value === 1)

р Добро пожаловать в первый раз

- else

р Не первое посещение:

session.js

let express = require('express');

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

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

let server = express();

server.set("view engine", "pug");

server.set("views", `./views`);

server.use(cookieParser());

server.use(session({secret: "Session data"}));

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

if(!req.session.page_views){

req.session.page_views = 0;

}

res.render("msg", {

value: ++req.session.page_views

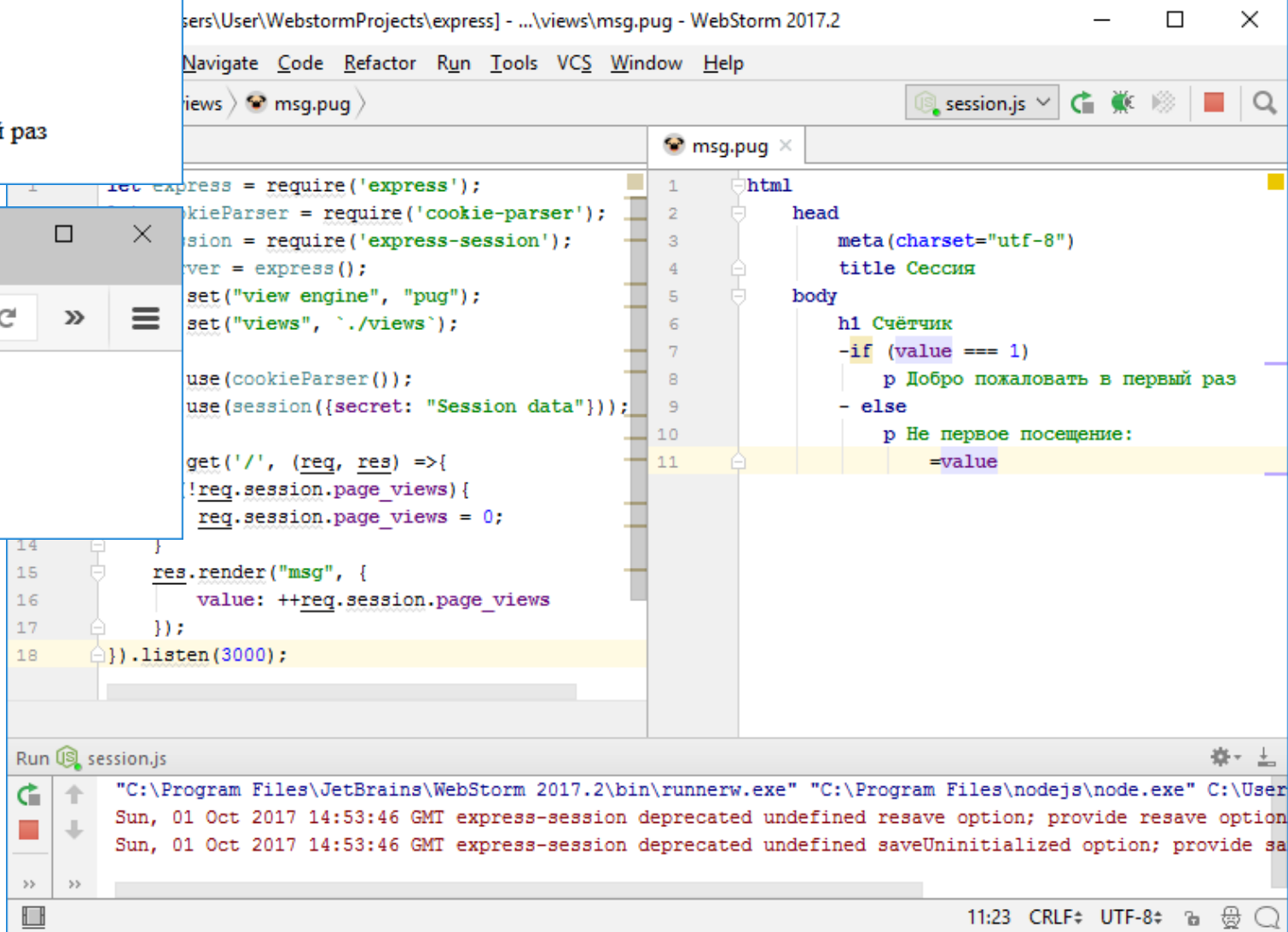
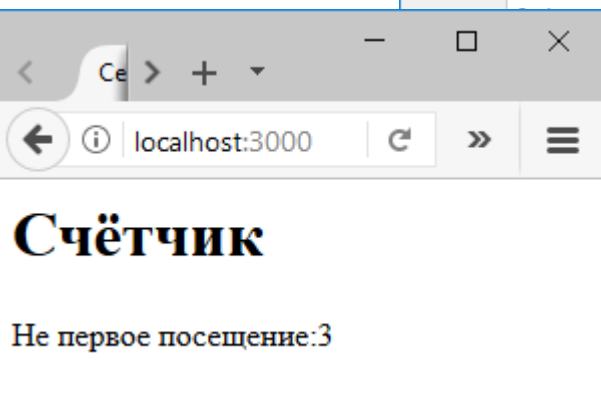
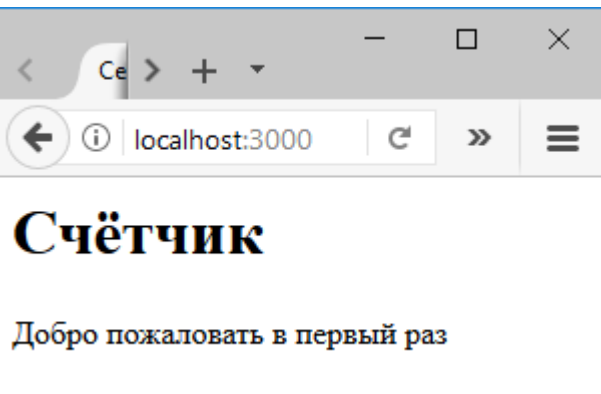
});

}).listen(3000);

<https://expressjs.com/en/resources/middleware/session.html>

Сессия (2)

33



Простая аутентификация (1)

34

login.js

```
let express = require('express');
let server = express();
let bodyParser = require('body-parser');
let session = require('express-session');
let cookieParser = require('cookie-parser');

server.set('view engine', 'pug');
server.set('views', './views');

server.use(bodyParser.json());
server.use(bodyParser.urlencoded({ extended: true }));
server.use(cookieParser());
server.use(session({secret: "Секретный ключ"}));

let users = [];

server.get('/login', (req, res)=>{
  res.render('login');
});

server.post('/login', (req, res)=>{
  if(!req.body.name || !req.body.password){
    res.status("400");
    res.end("Invalid details!");
  } else {
    for(user of users)
```

```
    if(user.name === req.body.name){
      res.render('login', {
        message: "Пользователь уже
        существует"
      });
      return;
    }
    let newUser = {name: req.body.name,
password: req.body.password};
    users.push(newUser);
    console.log(users);
    req.session.user = newUser;
    res.redirect('/protected_page');
  }
});
server.get('/protected_page', (req, res)=>{
  res.end(`Logged in for
  ${req.session.user.name}`);
}).listen(3000);
```

login.pug

```
html
  head
    meta(charset="utf-8")
    title Login
  body
    if(message)
      h4 #{message}
    form(action="/login" method="POST")
      input(name="name" required placeholder="Имя")
      input(name="password" type="password" required
placeholder="Пароль")
      button(type="Submit") Зарегистрировать
```

Простая аутентификация(2)

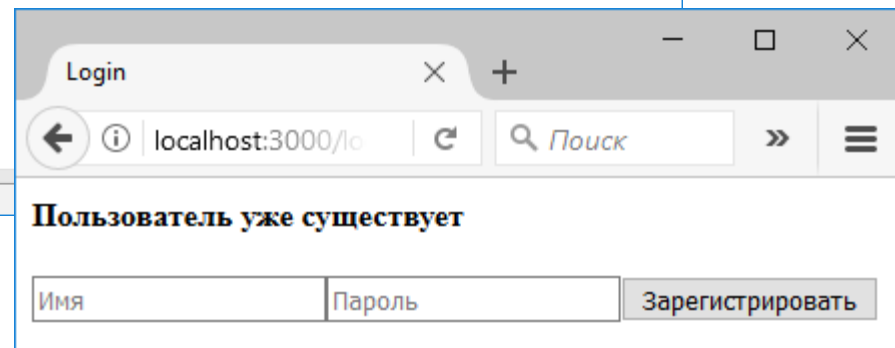
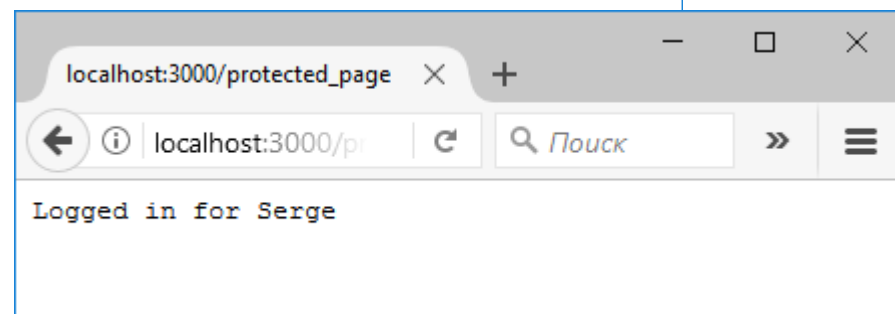
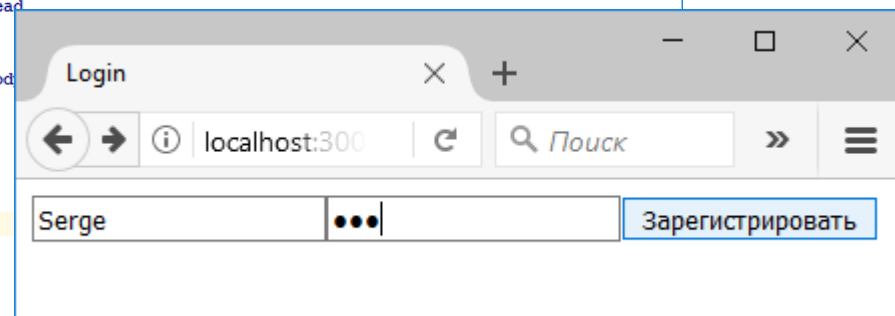
35

```
express - [C:\Users\User\WebstormProjects\express] - ...login.js - WebStorm 2017.2
File Edit View Navigate Code Refactor Run Tools VCS Window Help

express > login.js >
login.js x
login.pug x

1 let express = require('express');
2 let server = express();
3 let bodyParser = require('body-parser');
4 let session = require('express-session');
5 let cookieParser = require('cookie-parser');
6 server.set('view engine', 'pug');
7 server.set('views', './views');
8 server.use(bodyParser.json());
9 server.use(bodyParser.urlencoded({ extended: true }));
10 server.use(cookieParser());
11 server.use(session({secret: "Секретный ключ"}));
12 let users = [];
13 server.get('/login', (req, res)=>{
14   res.render('login');
15 });
16 server.post('/login', (req, res)=>{
17   if(!req.body.name || !req.body.password){
18     res.status("400");
19     res.end("Invalid details!");
20   } else {
21     for(user of users)
22       if(user.name === req.body.name){
23         res.render('login', {
24           message: "Пользователь уже существует"
25         });
26         return;
27       }
28     let newUser = {name: req.body.name, password: req.body.password};
29     users.push(newUser);
30     console.log(users);
31     req.session.user = newUser;
32     res.redirect('/protected_page');
33   }
34 });
35 server.get('/protected_page', (req, res)=>{
36   res.end('Logged in for ${req.session.user.name}');
37 }).listen(3000);

Unresolved function or method post()
```



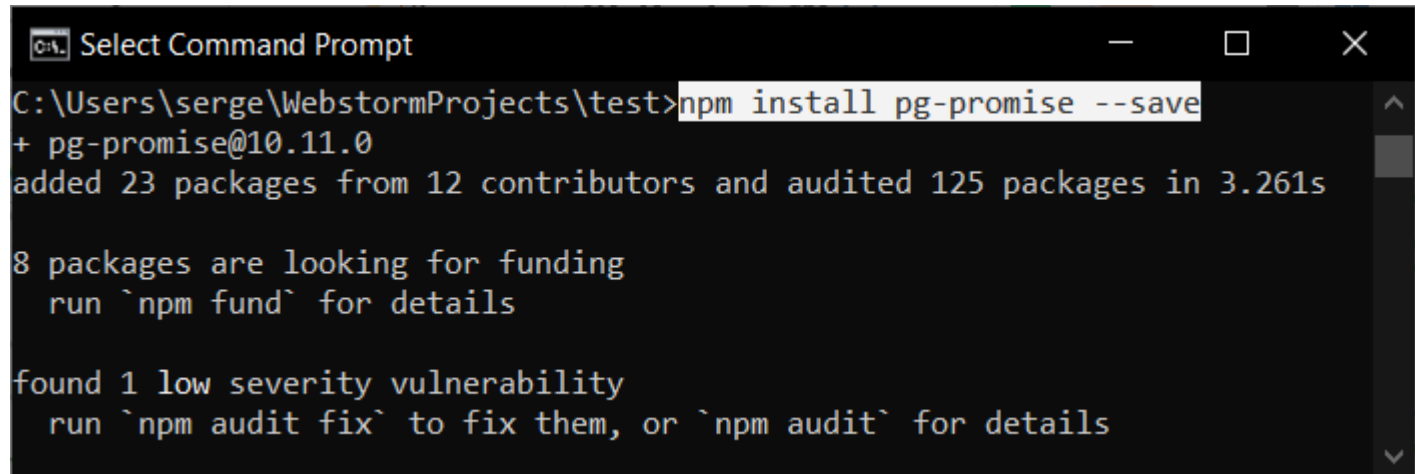
Интеграция express с PostgreSQL

36

```
let pgp = require("pg-promise")(/*options*/);
let db = pgp("postgres://username:password@host:port/database");

db.one("SELECT $1 AS value", 123)
  .then(function (data) {
    console.log("DATA:", data.value);
  })
  .catch(function (error) {
    console.log("ERROR:", error);
  });
```

npm install pg-promise --save



```
Select Command Prompt
C:\Users\serge\WebstormProjects\test>npm install pg-promise --save
+ pg-promise@10.11.0
added 23 packages from 12 contributors and audited 125 packages in 3.261s

8 packages are looking for funding
  run `npm fund` for details

found 1 low severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
```

<https://expressjs.com/ru/guide/database-integration.html>

Базовая безопасность в express

37

- **TLS** (Transport Layer Security)
 - Если приложение предназначено для работы с чувствительными данными
- **Helmet**
 - Помогает защитить приложение от некоторых широко известных веб-уязвимостей путем соответствующей настройки заголовков HTTP
- Безопасное использование **cookie**
 - Не используйте стандартные имена сеансовых cookie и соответствующим образом настройте опции защиты файлов cookie
 - `express-session`, `cookie-session`
- Введите **ограничение скорости** передачи данных во избежание атак методом грубого подбора сочетаний символов
- Используйте промежуточный обработчик `csurf` для защиты от подделки межсайтовых запросов (**CSRF**)
- Всегда применяйте фильтрацию и очистку пользовательского ввода в целях защиты от атак межсайтового скриптинга (**XSS**) и ввода ложных команд
- Обеспечьте защиту от атак **внедрения SQL-кода** с помощью параметризованных запросов
- Используйте инструмент **sqlmap**
- Используйте инструменты **nmap** и **sslyze** для проверки конфигурации
- Используйте **safe-regex**, чтобы убедиться в невосприимчивости регулярных выражений к атакам

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

<https://owasp.org/www-project-top-ten/>

Преимущества

- Почти стандарт для Node.js веб-промежуточное программное обеспечение
- Простой, минималистичный, гибкий и масштабируемый
- Быстрая разработка приложений
- Полностью настраиваемый
- Низкая кривая обучения
- Простая интеграция сторонних сервисов и промежуточного программного обеспечения
- В основном сосредоточен на браузерах, делая создание шаблонов и рендеринг практически готовыми функциями

Недостатки

- Организация должна быть очень четкой, чтобы избежать проблем при поддержании кода
- По мере увеличения размера вашей кодовой базы рефакторинг усложняется
- Нужно создать все endpoint

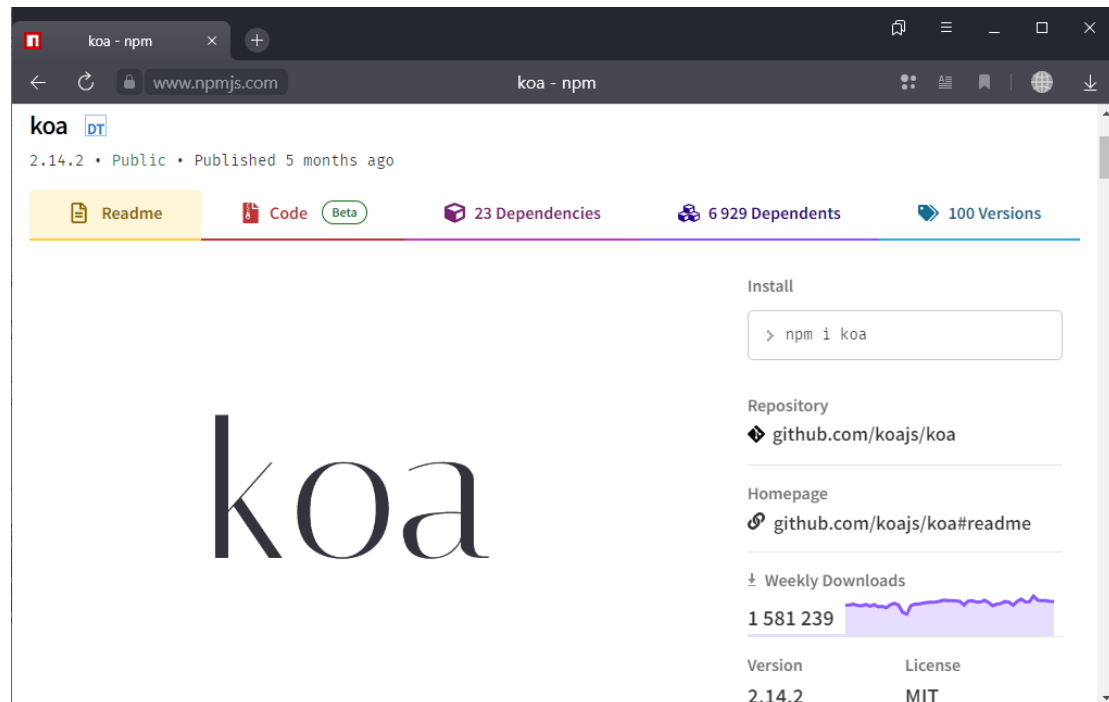
Сервер коа

- npm i koa
- npm i koa-pug
- npm i koa-static
- npm i koa-convert
- npm i koa-router

Особенности / модули	Коа	Express	Connect
Middleware	✓	✓	✓
Роутинг		✓	
Шаблоны		✓	
Отправка файлов		✓	

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

<https://koajs.com/>



The screenshot shows the npm package page for 'koa'. The page header includes the package name 'koa', version '2.14.2', and publication status 'Public • Published 5 months ago'. Below this are links for 'Readme', 'Code' (with a 'Beta' badge), '23 Dependencies', '6 929 Dependents', and '100 Versions'. The main content area features the 'koa' logo. On the right side, there is an 'Install' section with a command box containing 'npm i koa'. Below this are links for the 'Repository' (github.com/koajs/koa) and 'Homepage' (github.com/koajs/koa#readme). A 'Weekly Downloads' chart shows a steady increase in downloads, with a current count of '1 581 239'. At the bottom, a table lists the 'Version' as '2.14.2' and the 'License' as 'MIT'.

Version	License
2.14.2	MIT

Простейшее использование коа

app.js

```
const Koa = require('koa');
const app = new Koa();

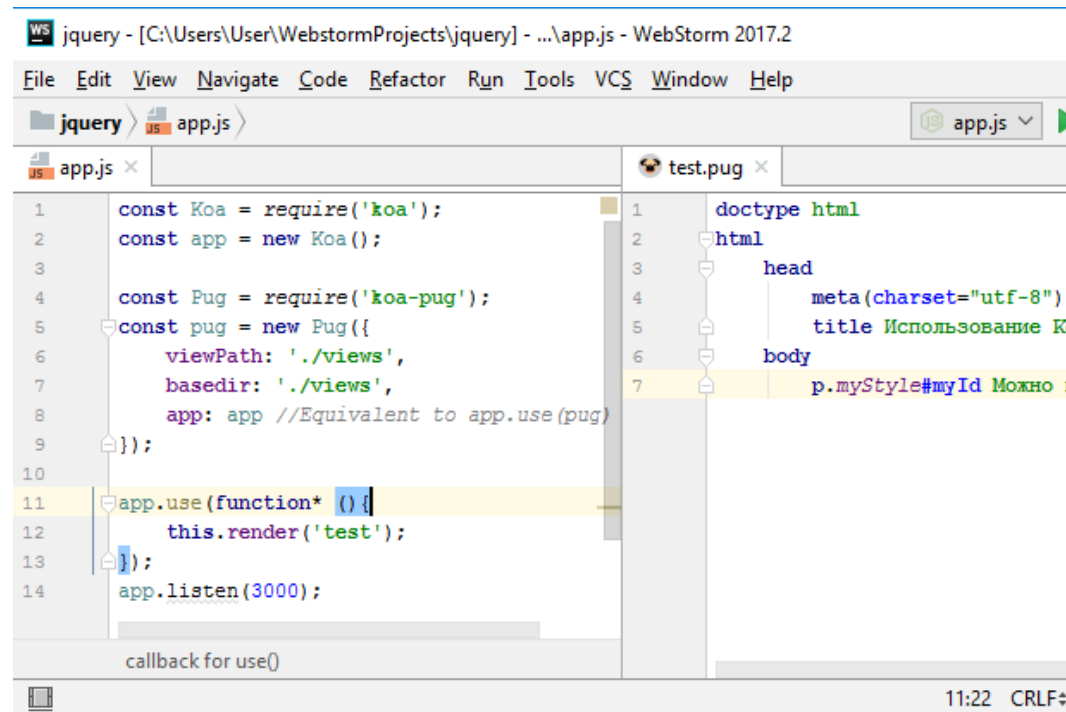
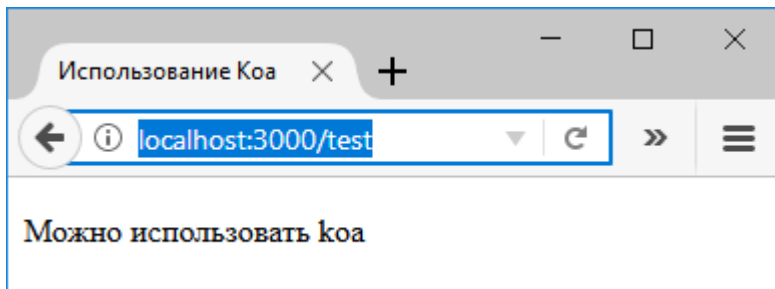
const Pug = require('koa-pug');
const pug = new Pug({
  viewPath: './views',
  basedir: './views',
  app: app //Equivalent to app.use(pug)
});

app.use(function* () {
  this.render('test');
});

app.listen(3000);
```

test.pug

```
doctype html
html
  head
    meta(charset="utf-8")
    title Использование Кoa
  body
    p.myStyle#myId Можно использовать коа
```



Использование коа с указанием путей

```
const Koa = require('koa');
const Router = require('koa-router');
const serve = require('koa-static-folder');
```

```
const app = new Koa();
const router = new Router();
```

```
const Pug = require('koa-pug');
const pug = new Pug({
  viewPath: './views',
  basedir: './views',
  app: app
});
```

```
router.get('/test', function (ctx, next) {
  ctx.response.body = pug.render('test');
});
```

```
app.use(serve('./public'));
app
  .use(router.routes())
  .use(router.allowedMethods());
app.listen(3000);
```

- **koa-router**
 - роутер Кoa
- **koa-static-folder**
 - задание статических папок
- **koa-pug**
 - использование PUG
- **router.get()**
 - задание относительных путей

Преимущества

- Коа улучшает совместимость, надежность и делает написание промежуточного программного обеспечения более простым
- Имеет большое количество полезных методов, но занимает мало места, так как в комплекте нет middleware
- Коа очень легкий
- Улучшенная обработка ошибок с помощью try/catch.
- Больше никаких обратных вызовов, облегчающих восходящий и нисходящий поток управления
- Более чистый, более читаемый асинхронный код

Недостатки

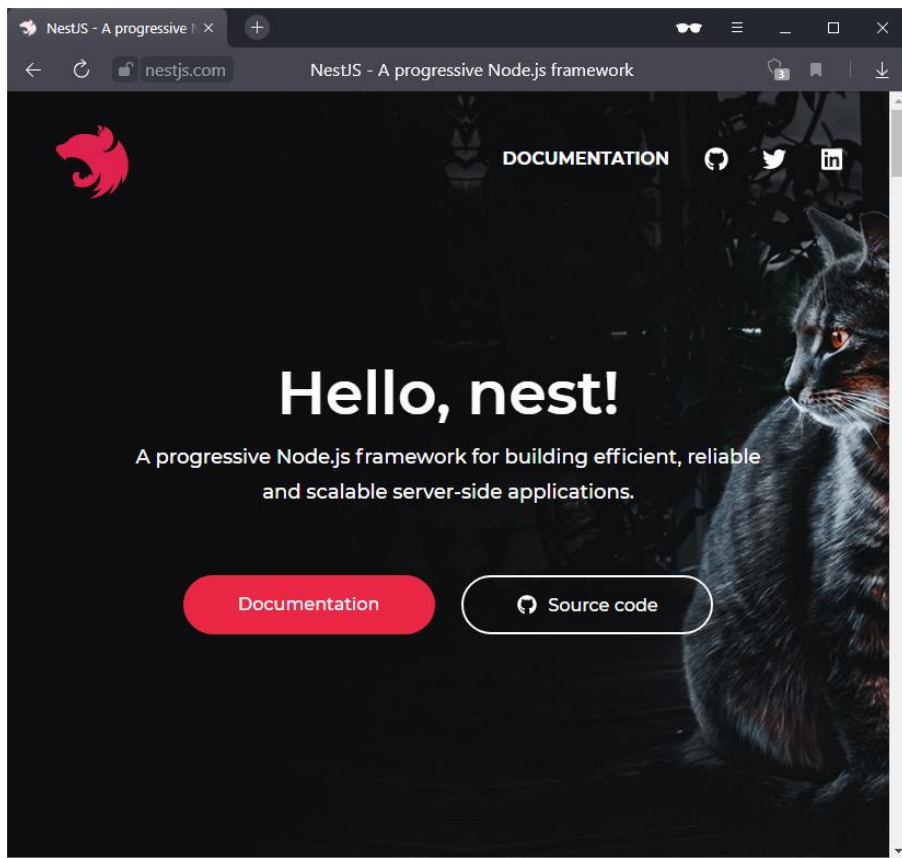
- Сообщество с открытым исходным кодом вокруг Коа невелико
- Не совместим с промежуточным программным обеспечением в стиле Express
- Коа использует генераторы, которые не совместимы ни с каким другим типом Node.js Middleware

NestJS – фреймворк разработки сервера

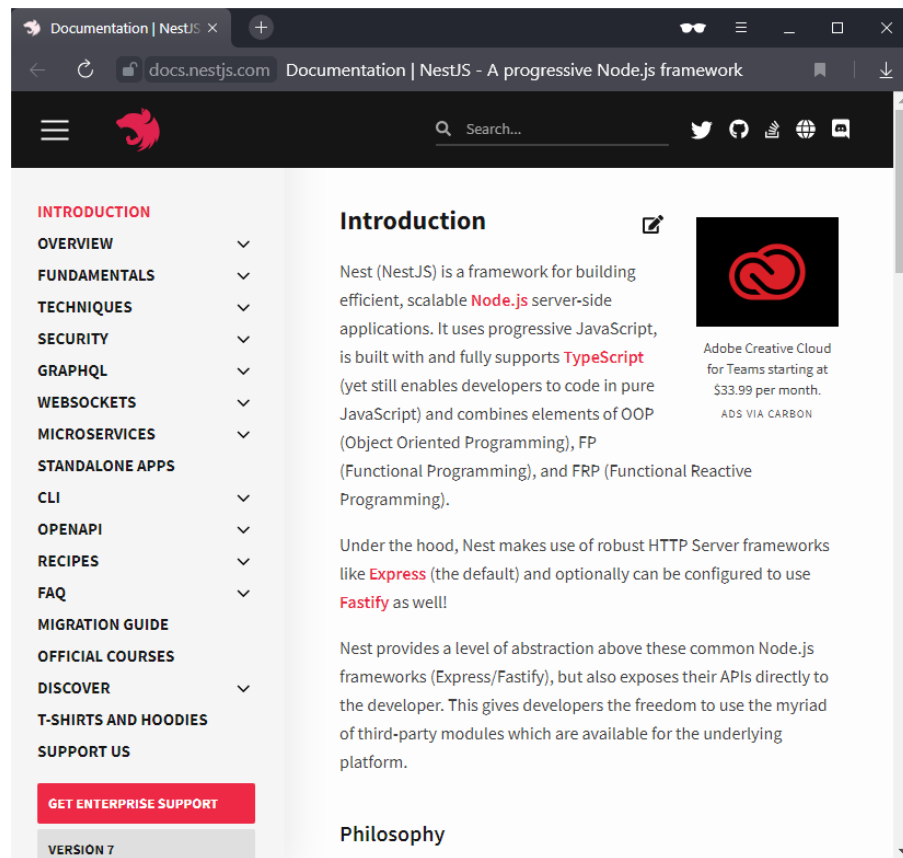
43

Live preview

<https://nestjs.com/>



<https://docs.nestjs.com/>



```
npm i @nestjs/cli -g  
nest new nestjs-getting-started
```

Быстрый старт на TypeScript
Но мы пойдём другим путём

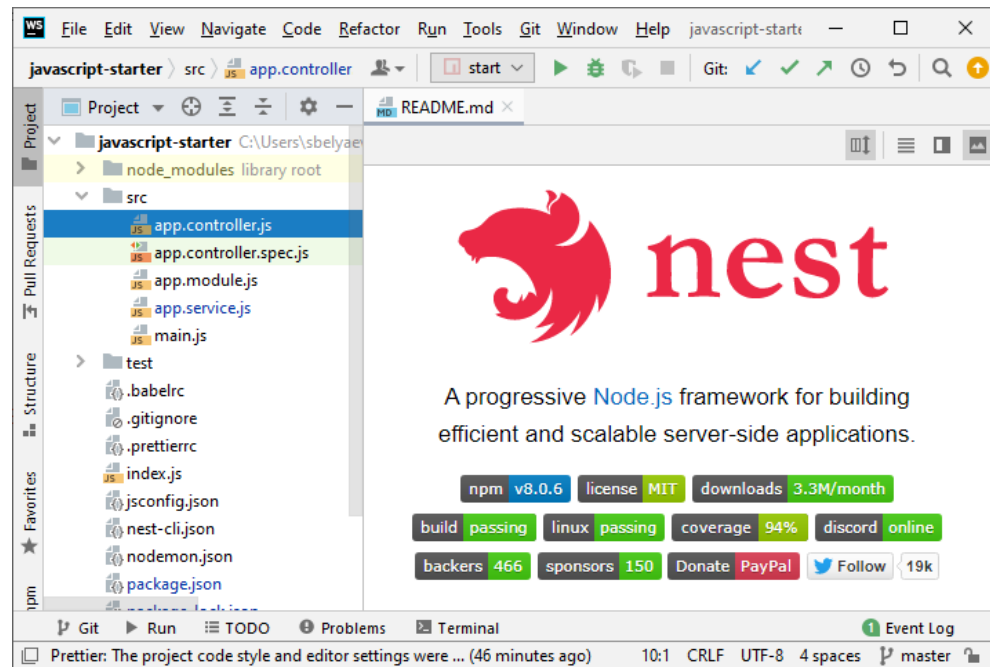
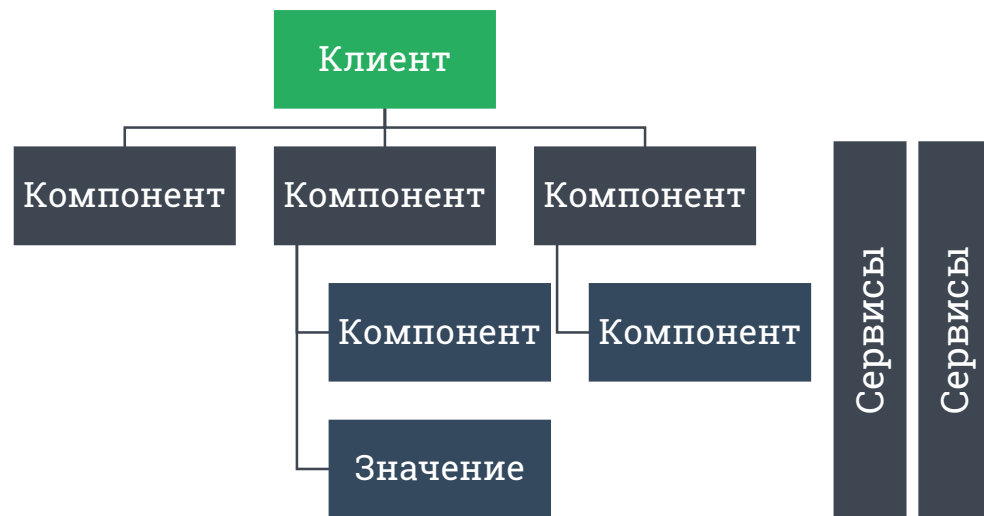
<https://github.com/juliandavidmr/awesome-nestjs>

Nest. Пример проекта

- **Модули** – блоки кода, решающие отдельные задачи
- **Компоненты** – способы объединения модулей
- **Сервисы** – компоненты, доступные в рамках всего приложения
- **Метаданные** – предоставление дополнительной информации

TypeScript

JavaScript



<https://github.com/nestjs/javascript-starter.git>

Nest. Приложение, модуль

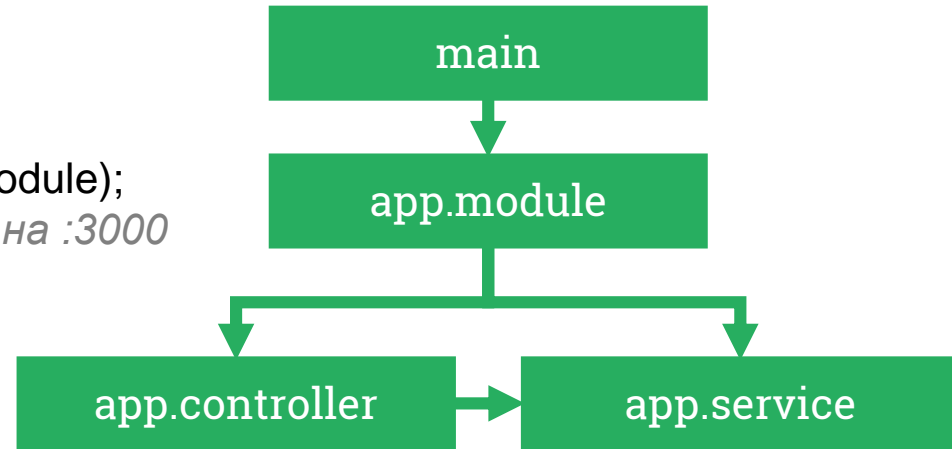
45

main.js

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
// Основное приложение
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3000); // Запуск слушателя на :3000
}
bootstrap(); // Запуск сервера
```

app.module.js

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
// Используемые модули
@Module({
  imports: [],
  controllers: [AppController], // Контроллеры
  providers: [AppService], // Сервисы
})
export class AppModule {}
```



Nest. Контроллер, сервис

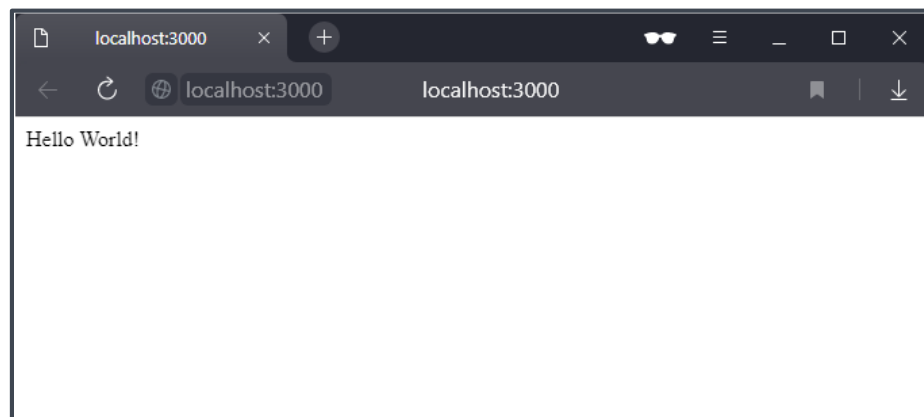
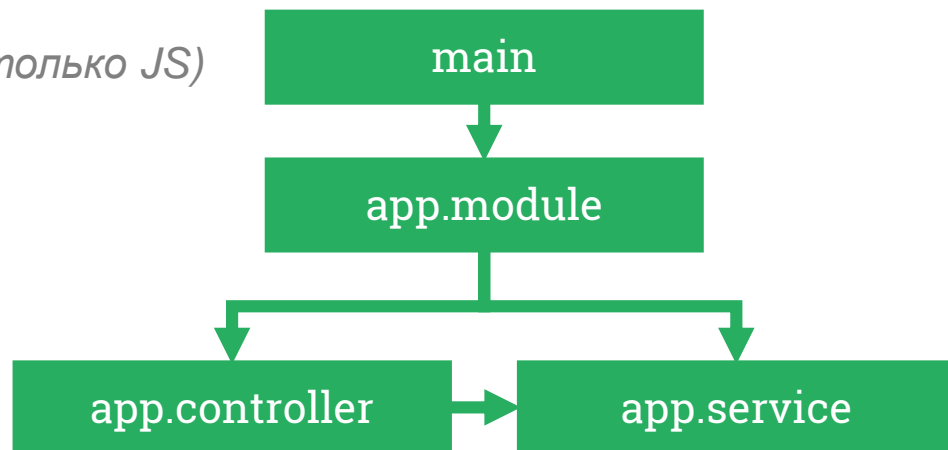
46

app.controller.js

```
import { Controller, Dependencies, Get } from '@nestjs/common';
import { AppService } from './app.service';
@Controller() // Контроллер
@Dependencies(AppService) // Использует (только JS)
export class AppController {
  constructor(appService) {
    this.appService = appService;
  }
  @Get() // Обращение к корню "/"
  getHello() { // "Произвольное" имя
    return this.appService.getHello();
  }
}
```

app.service.js

```
import { Injectable } from '@nestjs/common';
@Injectable() // Подключаемый
export class AppService {
  getHello() { // Вызываемый метод
    return 'Hello World!';
  }
}
```



CodeSandbox.io - Nest

47

The screenshot shows a CodeSandbox.io workspace titled "nest-typescript-starter". The interface includes a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a "src" directory with files like "app.controller.spec.ts", "app.controller.ts", "app.module.ts", "app.service.ts", and "main.ts", along with a "test" directory and configuration files like ".eslintrc.js", ".gitignore", ".prettierrc", "nest-cli.json", and "package-lock.json". The code editor displays the content of "main.ts", which imports "NestFactory" and "AppModule" from "@nestjs/core" and "../app.module" respectively. It defines an async "bootstrap" function that creates the application, listens on port 3000, and calls "bootstrap()". The terminal shows the command "start" being executed, with output indicating the application is successfully started. A preview window on the right displays "Hello World!".

```
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app.module';
3
4 async function bootstrap() {
5   const app = await NestFactory.create(AppModule);
6   await app.listen(3000);
7 }
8 bootstrap();
9
```

start

```
route +2ms
[Nest] 1133 - 09/12/2023, 4:00:06 PM LOG [NestApplication] Nest applicatio
n successfully started +4ms
```

Hello World!

<https://codesandbox.io/s/github/nestjs/typescript-starter/tree/master/>

Nest. Декораторы контроллера

- Создание «папки»

`@Controller("users")`

- теперь обращение не `http://localhost:3000`, а `http://localhost:3000/users` - все `@Get` будут в этой папке

- Создание запроса

`@Get("props")`

- теперь обращение не `http://localhost:3000`, а `http://localhost:3000/props` - переместили один `@Get`

- Примитивный тип возвращается «как есть», остальные – сериализуются в JSON

- Методы (импорт из `@nestjs/common`)

- `@Get()`, `@Post()`, `@Put()`, `@Delete()`, `@Patch()`, `@Options()`, `@Head()`, `@All()`

Параметры запроса

```
import { Controller, Dependencies, Bind, Get, Req } from '@nestjs/common';
import { AppService } from './app.service';
@Controller() // Контроллер
@Dependencies(AppService) // Использует
export class AppController {
  constructor(appService) {
    this.appService = appService;
  }
  @Get() // Обращение к корню "/"
  @Bind(Req()) // Подключение запросов (в TS выглядит по-другому!)
  getHello(req) { // "Произвольное" имя
    console.log(req) // Объект запроса
    return this.appService.getHello();
  }
}
```

Отличие для TS:

Bind не нужен

```
import { Response } from 'express';
getHello(@Req() request: Request): string
```

Параметры маршрута

```
import { Controller, Dependencies, Bind, Get, Param } from '@nestjs/common';
import { AppService } from './app.service';
@Controller() // Контроллер
@Dependencies(AppService) // Использует
export class AppController {
  constructor(appService) {
    this.appService = appService;
  }
  @Get(":id") // Обращение к корню "/123"
  @Bind(Param("id")) // Добавление параметра (в TS по-другому!)
  getHello(id) { // "Произвольное" имя
    console.log(id)
    return this.appService.getHello();
  }
}
```

Альтернатива

```
@Bind(Param())
getHello(param) {
  console.log(param.id)
  return this.appService.getHello();
}
```

Отличие для TS:

Bind не нужен

getHello(@Param('id') id): **string**

Дополнительные параметры (TS)

51

```
import { Controller, Get, HttpStatusCode, Header, Redirect, Res } from '@nestjs/common';  
import { Request } from 'express';  
import { Response } from 'express';
```

```
@Get()
```

```
@HttpStatusCode(204) // Код ответа
```

```
@Header('Cache-Control', 'none') // Заголовки
```

```
@Redirect('https://nestjs.com', 301) // Перенаправление
```

```
getHello(@Res() res: Response): string { // Ответ серверу
```

```
  console.log(res);
```

```
  return this.appService.getHello();
```

```
}
```

Промежуточные слои (Middleware)

52

Решаемы задачи:

- выполнение любого кода
- внесение изменений в объекты запроса и ответа
- завершение цикла запрос-ответ
- вызов следующей функции Middleware
- вызов функции `next()`, чтобы передать управление следующей функции Middleware

`my.middleware.js`

```
import { Injectable } from '@nestjs/common';
```

```
@Injectable()
```

```
export class MyMiddleware {  
  use(req, res, next) {  
    console.log('Request...');  
    next();  
  }  
}
```

Подключение на
следующем слайде



Подключение Middleware

53

```
import { Module } from '@nestjs/common';  
import { MyMiddleware } from '../common/middleware/my.middleware';  
import { UserModule } from '../users/user.module';
```

```
@Module({  
  imports: [UserModule], // Импорт модуля  
  controllers: [], // Контроллеры  
  providers: [] // Сервисы  
})  
export class AppModule {  
  configure(consumer) {  
    consumer  
      .apply(MyMiddleware) // Применение middleware  
      .forRoutes('users'); // Для маршрута  
  }  
}
```

Подключение шаблонов / pug

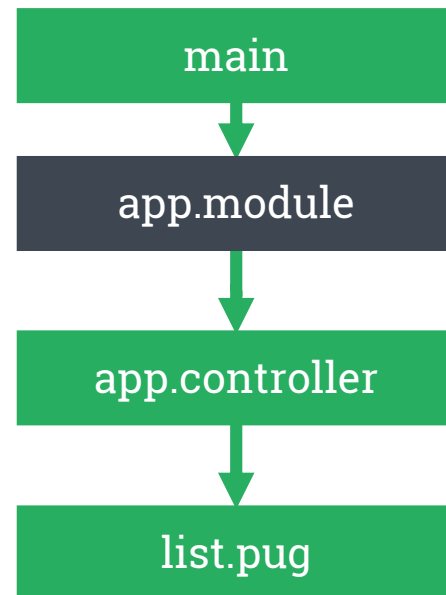
main.js

```
import { join } from 'path';
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
// Основное приложение
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  // Сообщим приложению, где искать views
  app.setBaseViewsDir(join(__dirname, '../views'));
  // И укажем, какой шаблонизатор использовать
  app.setViewEngine('pug');
  await app.listen(3000); // Запуск слушателя
}
bootstrap(); // Запуск сервера
```

Не забываем установить pug:

```
npm install --save pug
```

app.module в данном примере не изменяется (**app.service** не нужен)



Использование шаблонов / pug

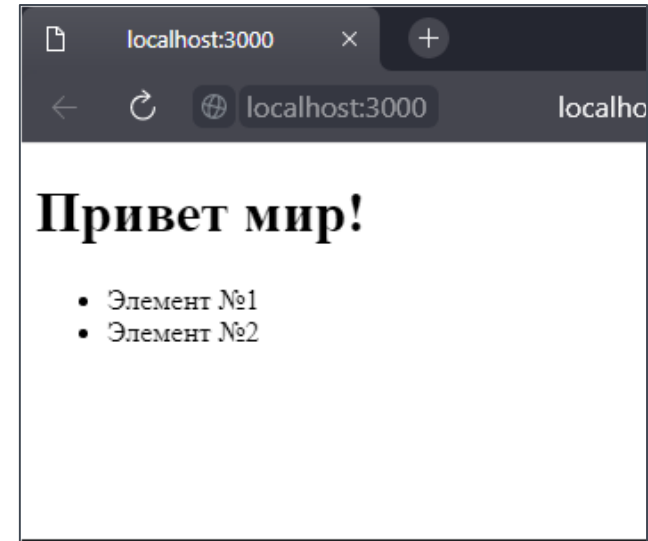
55

app.controller.js

```
import { Controller, Dependencies, Get, Render }  
    from '@nestjs/common';  
import { AppService } from './app.service';  
@Controller() // Контроллер  
@Dependencies(AppService) // Использует  
export class AppController {  
  constructor(appService) {  
    this.appService = appService;  
  }  
  @Get()  
  @Render("list") // Шаблон  
  getHello() {  
    return { list: ["Элемент №1", "Элемент №2"] };  
  }  
}
```

list.pug

```
doctype html  
html(lang="en")  
  head  
    meta(charset="UTF-8")  
  body  
    h1 Привет мир!  
    ul  
      each item in list  
        // #{...} — интерполяция  
        li #{item}
```



Дополнительные возможности

56

Exception
filters

Pipes

Guards

Interceptors

Custom
decorators

REST (Representational State Transfer)

Передача состояния представления

- Простота унифицированного интерфейса
- Открытость компонентов к возможным изменениям для удовлетворения изменяющихся потребностей
- Прозрачность связей между компонентами системы для сервисных служб
- Переносимость компонентов системы путем перемещения программного кода вместе с данными
- Надежность, выражающаяся в устойчивости к отказам на уровне системы при наличии отказов отдельных компонентов, соединений, или данных

Рой Филдинг

Требования к архитектуре REST

58

- Модель клиент-сервер
- Отсутствие состояния
- Кэширование
- Единообразие интерфейса
 - Идентификация ресурсов
 - Манипуляция ресурсами через представление
 - «Самоописываемые» сообщения (использование MIME)
 - Гипермедиа (HATEOAS), как средство изменения состояния приложения
- Слои
- Код по требованию

Методы REST

59

GET

получение
данных

POST

вставка
данных

PUT

обновление
или вставка

DELETE

удаление

REST (1)

60

app.js

```
let express = require('express');
let bodyParser = require('body-parser');
let cookieParser = require('cookie-parser');
let server = express();
```

```
server.use(cookieParser());
server.use(bodyParser.json());
server.use(bodyParser.urlencoded({ extended: true }));
```

```
let groups = require('./groups.js');
server.use('/groups', groups);
```

```
server.listen(3000);
```

groups.js

```
let express = require('express');
let router = express.Router();
let groups = [
  {id: 1, name: "5381", students: 15, rating: 4.1},
  {id: 2, name: "5303", students: 13, rating: 4.7},
  {id: 3, name: "5304", students: 14, rating: 5},
  {id: 4, name: "5382", students: 20, rating: 3.9},
  {id: 5, name: "5383", students: 14, rating: 4.9}
];
router.get('/', (req, res)=>{
  res.json(groups);
});
module.exports = router;
```

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET localhost:3000/groups
```

REST (2) - get

WS rest - [C:\Users\User\WebstormProjects\rest] - ...groups.js - WebStorm 2017.2

File Edit View Navigate Code Refactor Run Tools VCS Window Help

rest groups.js

app.js

```
1 let express = require('express');
2 let bodyParser = require('body-parser');
3 let cookieParser = require('cookie-parser');
4 let server = express();
5
6 server.use(cookieParser());
7 server.use(bodyParser.json());
8 server.use(bodyParser.urlencoded({ extended: true }));
9
10 let groups = require('./groups.js');
11 server.use('/groups', groups);
12
13 server.listen(3000);
```

Command Prompt

C:\Users\User\WebstormProjects>curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET localhost:3000/groups
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 249
ETag: W/"f9-IniXvdTP6kd3WRmOSYYtx8Ma5D4"
Date: Sun, 01 Oct 2017 16:08:15 GMT
Connection: keep-alive

[{"id":1,"name":"5381","students":15,"rating":4.1},{"id":2,"name":"5303","students":13,"rating":4.7},{"id":3,"name":"5304","students":14,"rating":5},{"id":4,"name":"5382","students":20,"rating":3.9},{"id":5,"name":"5383","students":14,"rating":4.9}]
C:\Users\User\WebstormProjects>

groups.js

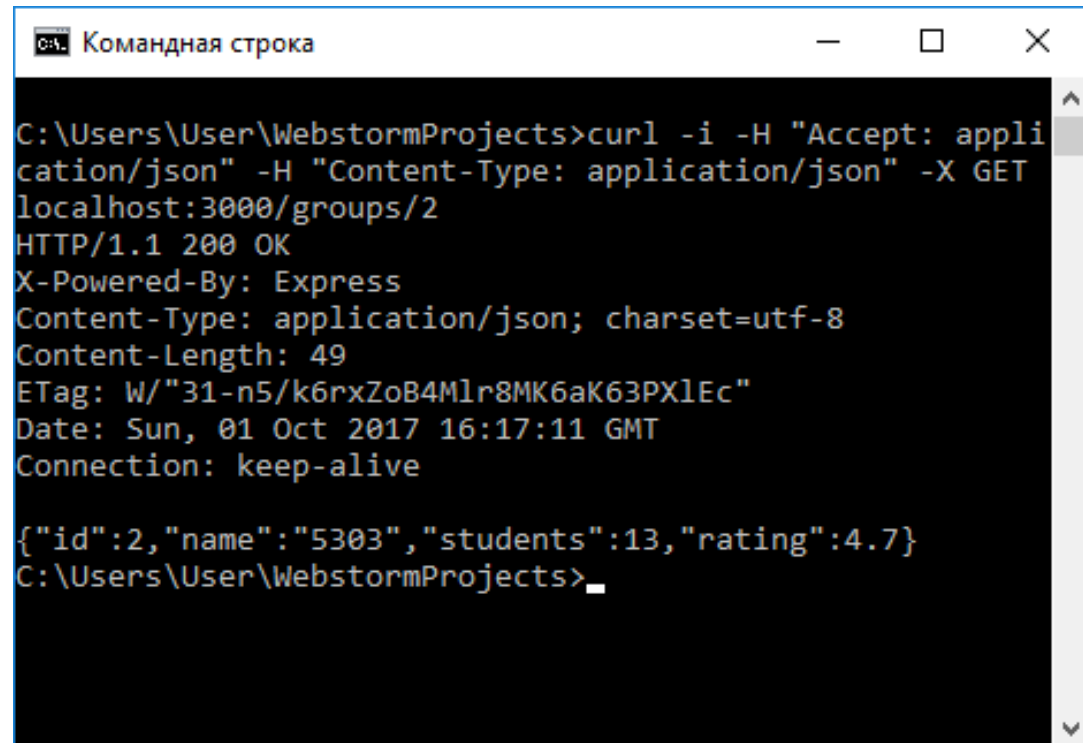
```
1 let express = require('express');
2 let router = express.Router();
3 let groups = [
4   {id: 1, name: "5381", students: 15, rating: 4.1},
5   {id: 2, name: "5303", students: 13, rating: 4.7},
6   {id: 3, name: "5304", students: 14, rating: 5},
7   {id: 4, name: "5382", students: 20, rating: 3.9},
8   {id: 5, name: "5383", students: 14, rating: 4.9}
9 ];
10 router.get('/', (req, res)=>{
11   res.json(groups);
12 });
13 module.exports = router;
```

13:25 CRLF UTF-8

REST (3) - get

62

```
router.get('/:id([0-9]{1,})', (req, res)=>{
  var group = groups.filter((g)=>{
    if(g.id == req.params.id)
      return true;
  });
  if(group.length == 1)
    res.json(group[0])
  else {
    res.status(404);
    res.json({message:
      "Not Found"});
  }
});
```



The screenshot shows a Windows Command Prompt window titled "Командная строка". The command entered is `curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET localhost:3000/groups/2`. The output shows an HTTP 200 OK response with headers including `X-Powered-By: Express`, `Content-Type: application/json; charset=utf-8`, `Content-Length: 49`, `ETag: W/"31-n5/k6rxZoB4Mlr8MK6aK63PX1Ec"`, `Date: Sun, 01 Oct 2017 16:17:11 GMT`, and `Connection: keep-alive`. The response body is a JSON object: `{"id":2,"name":"5303","students":13,"rating":4.7}`.

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET localhost:3000/groups/2
```

REST (4) - post

63

```
Командная строка

C:\Users\User\WebstormProjects>curl -X POST --data "name=666&students=9&rating=9.9" http://localhost:3000/groups

{"message":"New groupd created.","location":"/groups/6"}

C:\Users\User\WebstormProjects>
```

```
Командная строка

C:\Users\User\WebstormProjects>curl -i -H "Accept: application/json" -H "Content-Type: application/json" http://localhost:3000/groups/6
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 47
ETag: W/"2f-qXoyXoaSPk/JNf2bjzcY7Ks8VYQ"
Date: Sun, 01 Oct 2017 16:29:01 GMT
Connection: keep-alive

{"id":6,"name":"666","year":"9","rating":"9.9"}
C:\Users\User\WebstormProjects>
```

```
router.post('/', (req, res)=>{
  let body = req.body;
  if(!body.name ||
    !body.students.toString().match(/^([0-9]{1,})$/g) ||
    !body.rating.toString().match(/^([0-9]\.[0-9])$/g)) {
    res.status(400);
    res.json({message: "Bad Request"});
  } else {
    var newId = groups[groups.length-1].id+1;
    groups.push({
      id: newId,
      name: body.name,
      students: body.students,
      rating: body.rating
    });
    res.json({message: "New groupd created.", location: "/groups/" + newId});
  }
});
```

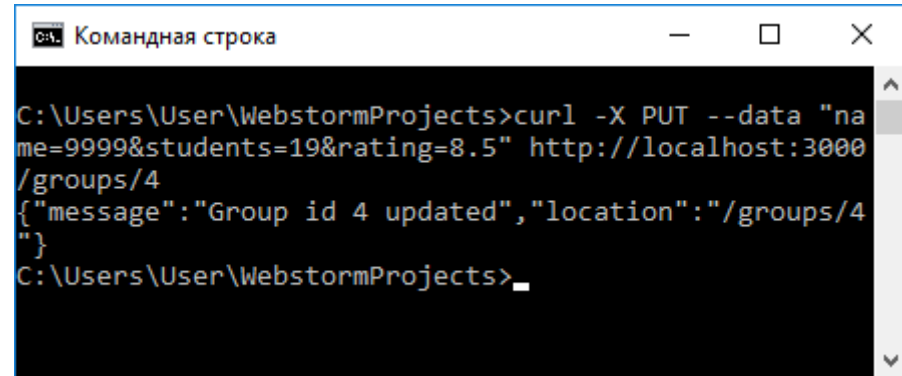
```
curl -X POST --data "name=666&students=9&rating=9.9" http://localhost:3000/groups
```

REST (5) - put

64

```
router.put('/:id', (req, res)=>{
  let body = req.body;
  if(!body.name ||
    !body.students.toString().match(/^([0-9]{1,})$/g) ||
    !body.rating.toString().match(/^([0-9]\.[0-9])$/g) ||
    !req.params.id.toString().match(/^([0-9]{1,})$/g)) {
    res.status(400);
    res.json({message: "Bad Request"});
  } else {
    var updateIndex = groups.map((group)=>{
      return parseInt(group.id);
    }).indexOf(parseInt(req.params.id));

    if(updateIndex === -1){
      var newId = groups[groups.length-1].id+1;
      groups.push({
        id: newId,
        name: body.name,
        students: body.students,
        rating: body.rating
      });
      res.json({message: "New group created.", location: "/groups/" + newId});
    } else {
      groups[updateIndex] = {
        id: req.params.id,
        name: body.name,
        students: body.students,
        rating: body.rating
      };
      res.json({message: `Group id ${req.params.id} updated`,
        location: "/groups/" + req.params.id});
    }
  }
});
```



```
Командная строка
C:\Users\User\WebstormProjects>curl -X PUT --data "name=9999&students=19&rating=8.5" http://localhost:3000/groups/4
{"message":"Group id 4 updated","location":"/groups/4"}
C:\Users\User\WebstormProjects>
```

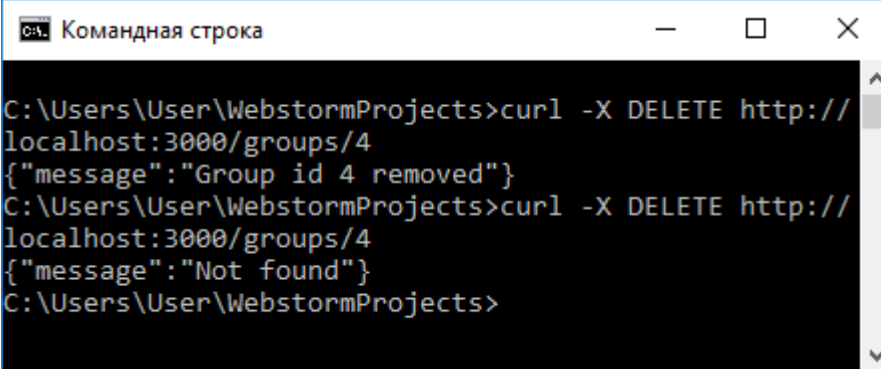
```
curl -X PUT --data "name=9999&students=19&rating=8.5" http://localhost:3000/groups/4
```


REST (6) - delete

65

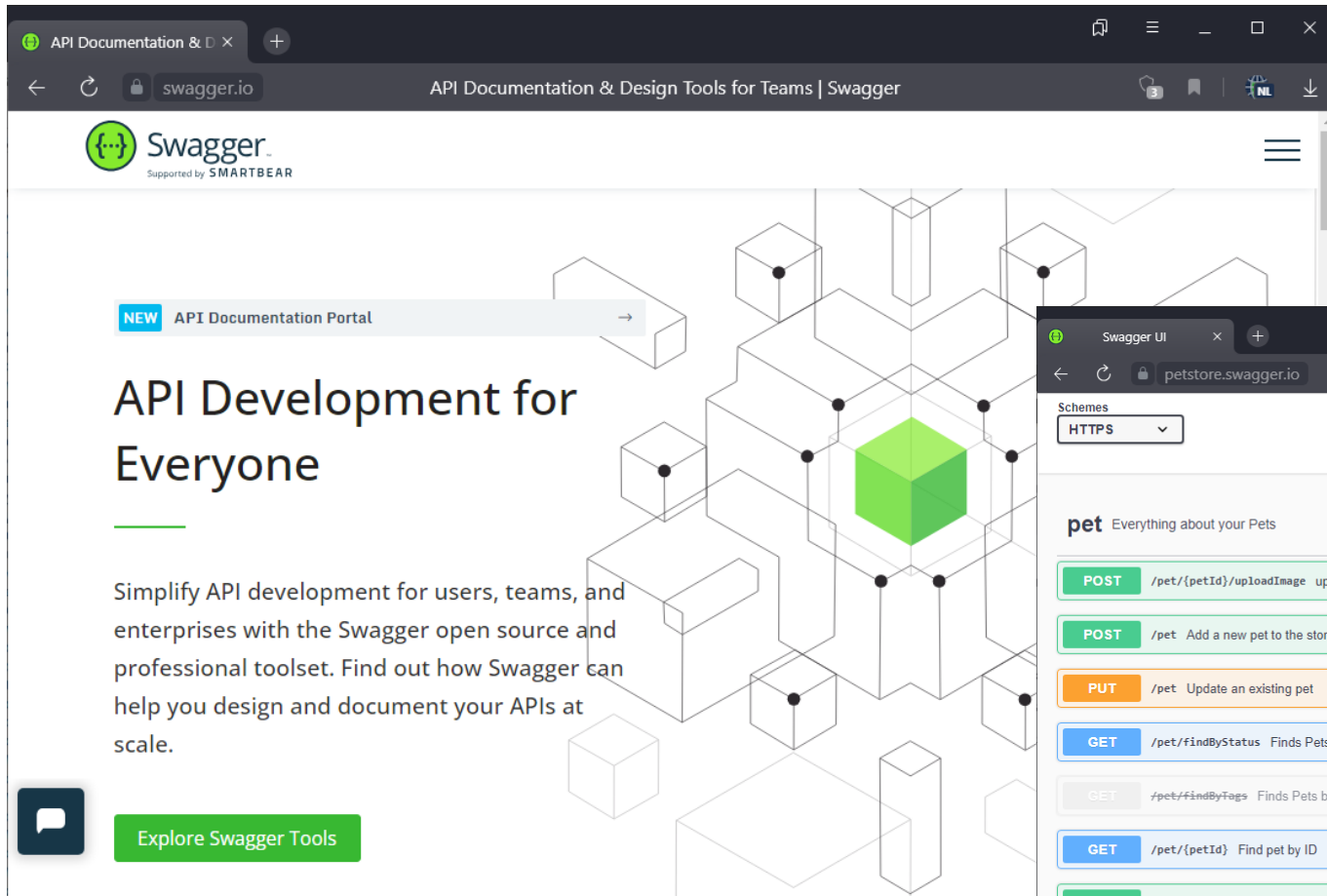
```
router.delete('/:id', (req, res)=>{
  var removeIndex = groups.map((group)=>{
    return parseInt(group.id);
  }).indexOf(parseInt(req.params.id));

  if(removeIndex === -1){
    res.json({message: "Not found"});
  } else {
    groups.splice(removeIndex, 1);
    res.send({message: `Group id ${req.params.id} removed`});
  }
});
```

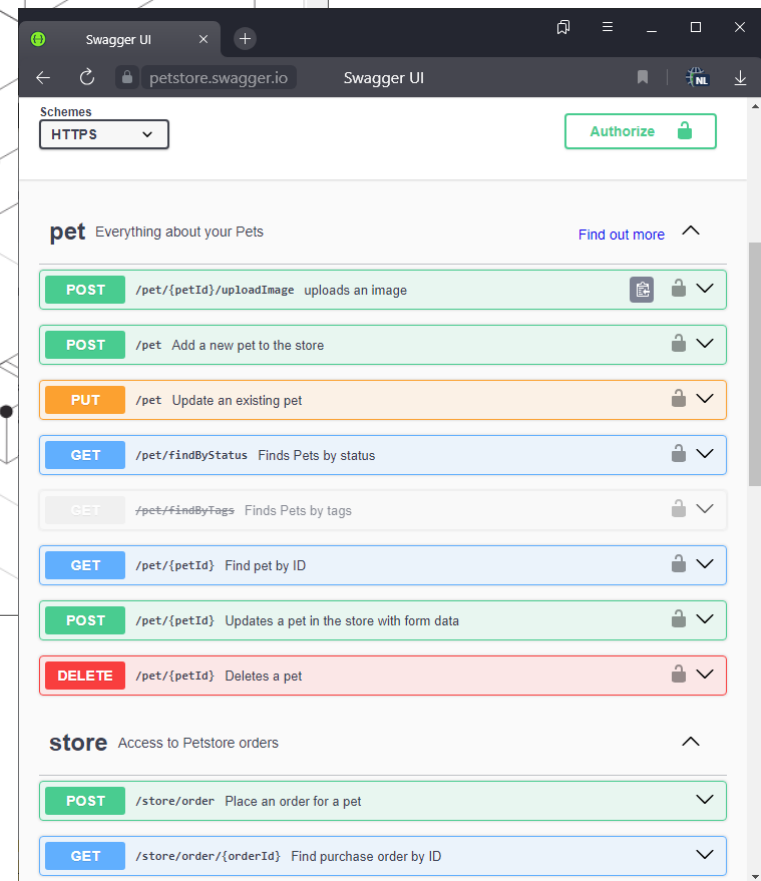


```
Командная строка
C:\Users\User\WebstormProjects>curl -X DELETE http://localhost:3000/groups/4
{"message":"Group id 4 removed"}
C:\Users\User\WebstormProjects>curl -X DELETE http://localhost:3000/groups/4
{"message":"Not found"}
C:\Users\User\WebstormProjects>
```

curl -X DELETE http://localhost:3000/groups/4



<https://swagger.io/>

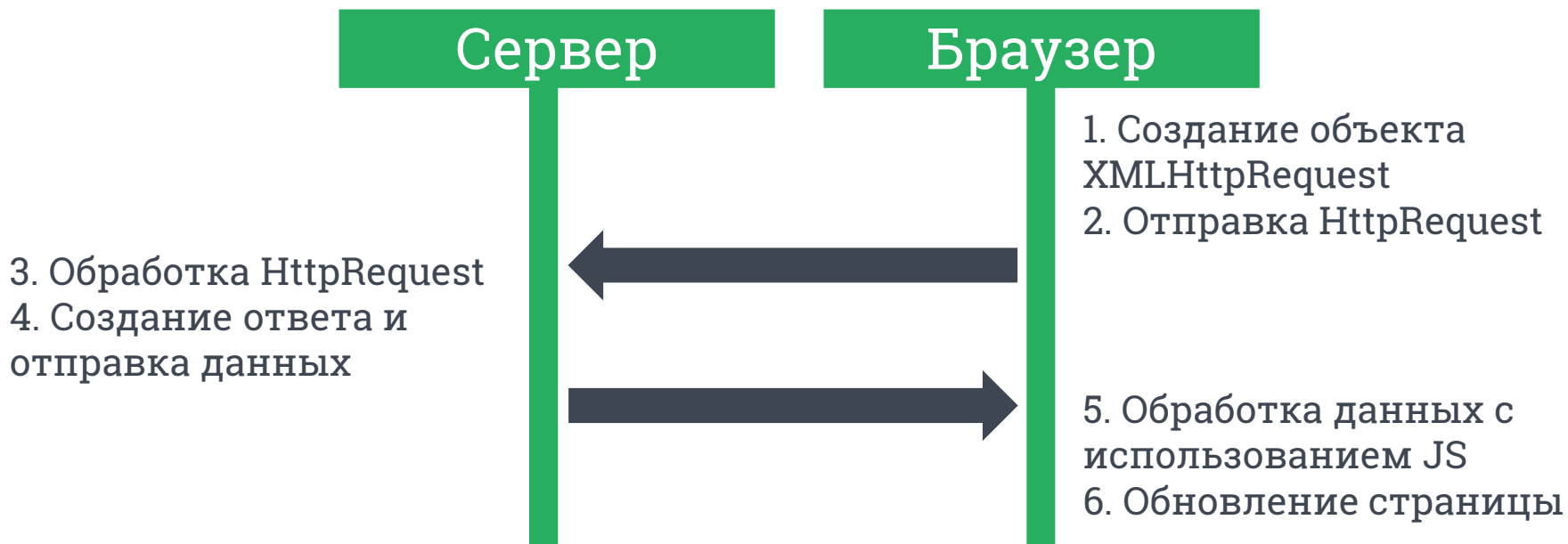


<https://petstore.swagger.io/>

Понятие AJAX

67

- AJAX ⇔ **A**synchronous **J**avaScript **A**nd **X**ML
- AJAX использует комбинацию
 - встроенного в браузер объекта XMLHttpRequest, чтобы запросить данные с сервера
 - JavaScript и HTML DOM для отображения и использования данных
- AJAX позволяет отправлять на сервер асинхронные запросы для обновления части страницы, а не страницы целиком



Пример XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<training_program>
  <content> <!-- Комментарий -->
    Содержание программы
  </content>
  <references>
    <ref url="https://www.tutorialspoint.com/expressjs/" />
    <ref url="https://www.w3schools.com/" />
  </references>
  <units>
    <unit type="rest">
      <description/>
      <get/>
      <post/>
      <put/>
      <delete/>
    </unit>
    <unit type="OAuth">
      <description/>
      <mail_ru></mail_ru>
      <vk_com></vk_com>
    </unit>
    <unit type="Ajax"></unit>
  </units>
</training_program>
```

<https://www.w3schools.com/xml/>

- DOM
- DTD, XML Schema
- Parser (DOM, SAX, StAX)
- XSLT
- XPath

AJAX (1) – сервер

69

app.js

```
let express = require("express");
let server = express();
let routes = require("./routes");
server.use('/public', express.static('public'));
server.set("view engine", "pug");
server.set("views", `./views`);

server.use("/", routes);

server.listen(3000);
```

routes.js

```
let express = require("express");
let router = express.Router();
let groups = [
  {num: "5381", amount: "15"},
  {num: "5303", amount: "13"},
  {num: "5304", amount: "14"},
  {num: "5382", amount: "20"},
  {num: "5383", amount: "14"}];

router.get("/", (req, res, next) => {
  res.render("index", {
    groups: groups
  });
  next();
});

router.get("/group/:num", (req, res, next) => {
  let number = req.params.num;
  for (value of groups)
    if (value.num === number)
      res.end(JSON.stringify(value));
  next();
});

router.get("/group/*", (req, res) => {
  res.end(JSON.stringify({error: 'No such group'}));
});

module.exports = router;
```

AJAX (2) – представление

70

index.pug

```
doctype html
html
```

```
  head
```

```
    meta(charset="utf-8")
    link(href='/public/w3.css', rel='stylesheet')
    script(src="/public/ajax.js")
    title Ajax
```

```
  body
```

```
    h1 Демонстрация Ajax
```

```
    - for(var key in groups)
      -value=groups[key]
```

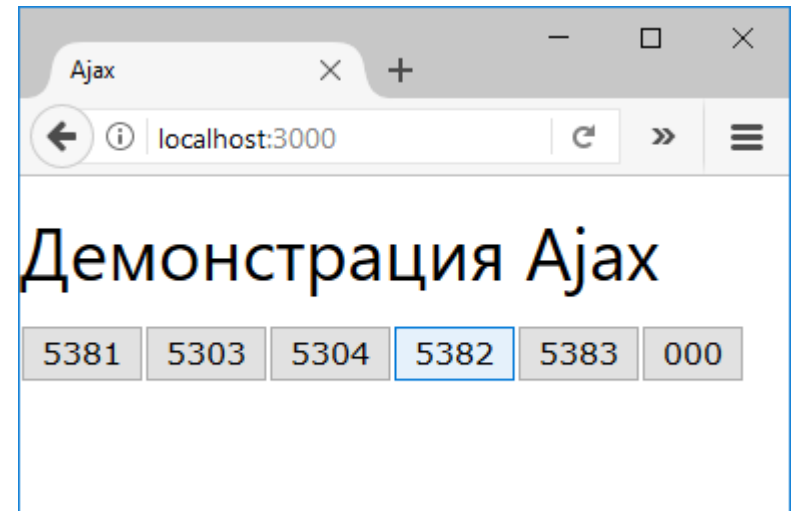
```
      button(id=value.num onclick='load(this)')=value.num
```

```
    button(id="000" onclick='load(this)') 000
```

```
    div
```

```
      span#title(style="visibility: hidden;") Количество студентов в группе
```

```
      span#amount.w3-red
```



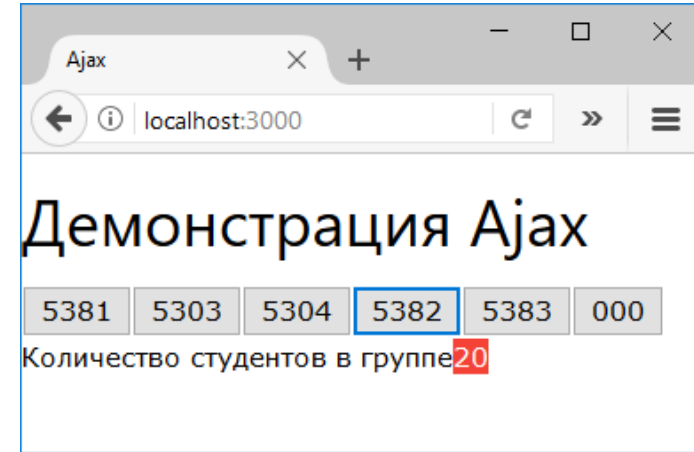
AJAX (3) – клиентский JS

71

ajax.js

```
function load(button) {
    let id = button.id;
    let title = document.getElementById("title");
    let amount = document.getElementById("amount");
    title.style.visibility = "visible";
    callAjax(id, (response)=>{
        try {
            let group = JSON.parse(response);
            if(group.error) {
                title.style.visibility = "collapse";
                amount.innerHTML = group.error;
            } else
                amount.innerHTML = group.amount;
        } catch (e) {
            title.style.visibility = "collapse";
            amount.innerHTML = "Ошибка: " + e;
        }
    });
}

function callAjax(id, callback) {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200)
            callback(this.responseText);
    };
    xhttp.open("GET", `/group/${id}`, true);
    xhttp.send();
}
```



События по ходу обработки запроса ⁷²

- **readystatechange** – изменение состояния обработки
- **loadstart** – запрос начат
- **progress** – браузер получил очередной пакет данных, можно прочитать текущие полученные данные в `responseText`
- **abort** – запрос был отменён вызовом `abort()`
- **error** – произошла ошибка
- **load** – запрос был успешно (без ошибок) завершён
- **timeout** – запрос был прекращён по таймауту
- **loadend** – запрос был завершён (успешно или неуспешно)

onreadystatechange	ontimeout
onerror	onload
onprogress	onabort
onloadstart	onloadend

Методы объекта XMLHttpRequest

73

- **open()**
 - open(method, URL)
 - open(method, URL, async)
 - open(method, URL, async, userName)
 - open(method, URL, async, userName, password)
- **send()**
- **abort()**
- **setRequestHeader(name, value)**
 - устанавливает заголовок name запроса со значением value
- **getAllResponseHeaders()**
 - возвращает строку со всеми HTTP-заголовками ответа сервера.
- **getResponseHeader(headerName)**
 - возвращает значение заголовка ответа сервера с именем headerName.

Свойства объекта XMLHttpRequest

74

- **onreadystatechange**
 - ссылается на функцию-обработчик состояний запроса
- **readyState**
 - номер состояния запроса от 0 до 4 (UNSENT, OPENED, HEADERS_RECEIVED, LOADING, DONE)
 - Используйте только 4 ("DONE")
- **responseText**
 - текст ответа сервера (при readyState=4)
- **responseXML**
 - ответ сервера в виде XML (при readyState=4)
- **status**
 - для HTTP-запросов – статусный код ответа сервера:
200 - OK, 404 - Not Found, и т.п.
- **statusText**
 - текстовая расшифровка status, например "Not Found" или "OK"

AJAX – GET

routes.js

```
let express = require("express");
let router = express.Router();
let url = require("url");
let groups = [
  {num: "5381", amount: "15"},
  {num: "5303", amount: "13"},
  {num: "5304", amount: "14"},
  {num: "5382", amount: "20"},
  {num: "5383", amount: "14"}];

router.get("/", (req, res, next) => {
  res.render("indexParam", {
    groups: groups
  });
  next();
});

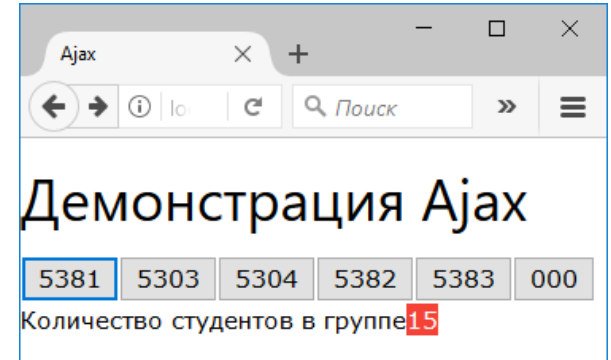
router.get("/group/", (req, res) => {
  let urlParts = url.parse(req.url, true);
  let number = urlParts.query.num;
  let found = false;
  for (value of groups) {
    if(value.num === number) {
      res.end(JSON.stringify(value));
      found = true;
      break;
    }
  }
  if(!found)
    res.end(JSON.stringify({error: 'No such group'}));
});
module.exports = router;
```

ajax.js

75

```
function load(button) {
  let id = button.id;
  let title = document.getElementById("title");
  let amount = document.getElementById("amount");
  title.style.visibility = "visible";
  callAjax(id, (response)=>{
    try {
      let group = JSON.parse(response);
      if(group.error) {
        title.style.visibility = "collapse";
        amount.innerHTML = group.error;
      } else
        amount.innerHTML = group.amount;
    } catch (e) {
      title.style.visibility = "collapse";
      amount.innerHTML = "Ошибка: " + e;
    }
  });
}

function callAjax(id, callback) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200)
      callback(this.responseText);
  };
  xhttp.open("GET", `/group/?num=${id}`, true);
  xhttp.send();
}
```



AJAX: GET vs. POST

GET

```
function callAjax(id, callback) {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200)  
            callback(this.responseText);  
    };  
    xhttp.open("GET", `/group/?num=${id}`, true);  
    xhttp.send();  
}
```

Модули на сервере для разбора параметров:

- url

Пример разбора:

```
let urlParts = url.parse(req.url, true);  
let number = urlParts.query.num;
```

1. Проще и быстрее

Несколько параметров передаются через &, например:
a=1234&b=123&c=321

POST

```
function callAjax(id, callback) {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200)  
            callback(this.responseText);  
    };  
    xhttp.open("POST", "/group/", true);  
    xhttp.send(`num=${id}`);  
}
```

Модули на сервере для разбора параметров:

- body-parser

Пример разбора:

```
let number = req.body.num;
```

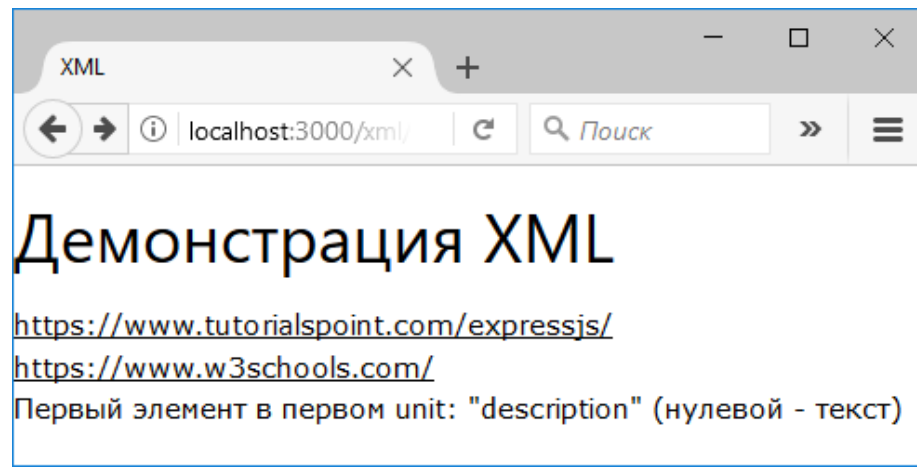
1. Не кэшируется
2. Отправка больших объёмов данных
3. Ввод от пользователя

Запрос XML с сервера

77

```
function load() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200)  
            useXml(this.responseXML);  
    };  
    xhttp.open("GET", "/public/demo.xml", true);  
    xhttp.send();  
}  
  
function useXml(xml) {  
    let div = document.getElementById("data");  
    let refList = xml.getElementsByTagName("ref");  
    div.innerHTML = "";  
    for(ref of refList)  
        div.innerHTML +=  
`<a href="${ref.attributes[0].nodeValue}">${ref.attributes[0].nodeValue}</a><br>`;  
    let unitList = xml.getElementsByTagName("unit");  
    if(unitList.length > 0)  
        div.innerHTML +=  
`Первый элемент в первом unit: "${unitList[0].childNodes[1].nodeName}" (нулевой - текст)`;  
}
```

Использован demo.xml со
слайда «Пример XML»



fetch – get

78

```
let response = await fetch(  
  'http://localhost:3000/', // Адрес запроса  
  {  
    method: 'GET' // Параметры запроса  
  });  
let text = await response.text(); // Текст  
console.log(text); // HTML-ответ
```

```
fetch(  
  'http://localhost:3000/',  
  {  
    method: 'GET'  
  })  
  .then(response=>response.text())  
  .then(result=>console.log(result))
```

response.ok == true // код HTTP-статуса в диапазоне 200-299
response.status – код ответа
response.headers – заголовки ответа

Response предоставляет несколько методов для доступа к телу ответа в различных форматах:

- **response.text()** – читает ответ и возвращает как обычный текст
- **response.json()** – декодирует ответ в формате **JSON**
- **response.formData()** – возвращает ответ как объект **FormData**
- **response.blob()** – возвращает объект как **Blob**
- **response.arrayBuffer()** – возвращает ответ как **ArrayBuffer**

fetch – post

```
let data = {}
let response = await fetch(
  'http://localhost:3000/', // Адрес запроса
  {
    method: 'POST', // Метод
    headers: { // Заголовки
      'Content-Type': 'application/json;charset=utf-8'
    },
    body: JSON.stringify(data) // Тело запроса
  }
);
console.log(response.headers.get('Content-Type')) // Тип ответа
let json = await response.json(); // Получение json
console.log(json); // HTML-ответ
```

Можно выбрать только один метод `text()` либо `json()`.
Повторный вызов приведёт к ошибке

CORS-ошибки

- No 'Access-Control-Allow-Origin' header is present on the requested resource.
- Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <https://example.com/>
- Access to fetch at 'https://example.com' from origin 'http://localhost:3000' has been blocked by CORS policy.

// Настройка Express

```
server.use((req, res, next) => {
  // Указываем явно откуда придёт запрос либо пишем *
  res.header('Access-Control-Allow-Origin', 'https://habr.com');
  // Origin, X-Requested-With, Content-Type, Асепт - варианты заголовков
  res.header('Access-Control-Allow-Headers', 'Content-Type');
  res.header('Access-Control-Allow-Methods', 'POST, GET, OPTIONS, DELETE');
  next();
})
```

CORS-запрос:

- к другому протоколу
- к другому порту
- к другому домену

<https://habr.com/ru/company/macloud/blog/553826/>

// Запрос из клиента

```
fetch(
  'http://localhost:3000',
  {
    method: 'GET',
    headers: {}
  })
.then(response=>response.text())
.then(result=>console.log(result))
```


Вопросы для самопроверки

81

- Что такое сервер Express? Какие его основные возможности?
- Как сделать маршрутизацию на Express?
- Для чего нужно Middleware? Какое оно бывает?
- Как обрабатывать ошибки в Express?
- Зачем нужны PUG? EJS? Какие у них отличия?
- Как можно получить данные из html-формы на сервере?
- Какой жизненный цикл у cookie?
- Как создать cookie на сервере?
- Как создать сессию? Что для этого нужно?
- Чем отличается Кoa от Express?
- Что такое NestJS? На каких языках он работает?
- Какая базовая архитектура проекта в NestJS?
- Для чего используются декораторы в NestJS? Какие бывают?
- Что такое REST? Какие требования к REST?
- Какие http-методы используют в REST?
- Что такое AJAX? Для чего он используется?
- В чём отличие отправки post и get запросов?
- Для чего нужен XMLHttpRequest? fetch?
- Что такое CORS? Как с ним «бороться»?