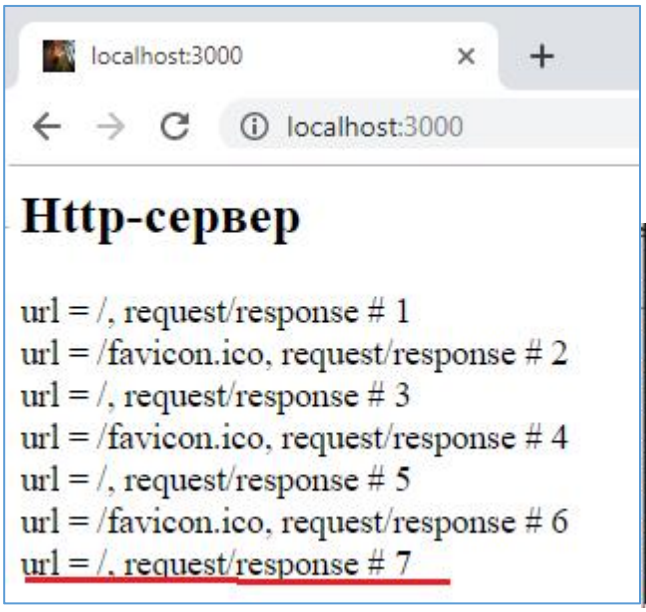
Node.js-HTTP-сервер

1. **HTTP-сервер:** серверная часть web-приложения
2. **HTTP-сервер:** простейший сервер

```
var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let s = '';

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

        console.log(`request url: ${req.url}, # `, ++k);

        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать заголовок

        res.write('<h2>Http-сервер</h2>');                                    // отправить порцию

        res.end( s += `url = ${req.url}, request/response # ${k}<br />`);     //отправить последнюю порцию

}).listen(3000);
```




```
D:\PSCA\Lec06>node 06-02
request url: /, #  1
request url: /favicon.ico, #  2
request url: /, #  3
request url: /favicon.ico, #  4
request url: /, #  5
request url: /favicon.ico, #  6
request url: /, #  7
request url: /favicon.ico, #  8
```

3. **HTTP-сервер:** простейший сервер

```
var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let s = '';

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

            console.log(`request url: ${req.url}, # `, ++k);

            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать

            res.write('<h2>Http-сервер</h2>');                                    // отправит

            s += `url = ${req.url}, request/response # ${k}<br />`;

            res.end(s);                                                           //отправить


});

server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

```
D:\PSCA\Lec06>
D:\PSCA\Lec06>node 06-03.js
server.listen(3000): error:  EADDRINUSE

D:\PSCA\Lec06>
```

4. **HTTP-сервер:** простейший сервер

```
//--------------- 06-04.js -------------------------------------

var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let s = '';

let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать
            res.write('<h2>Http-сервер</h2>');                                    // отправи
            s += `url = ${req.url}, request/response # ${k}<br />`;
            res.end(s);
};

let server = http.createServer(http_handler);

server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

5. **HTTP-сервер:** простейший сервер

```
//-------------- 06-05.js -------------------------------------

var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let s = '';

let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //з
            res.write('<h2>Http-сервер</h2>');                                    //
            s += `url = ${req.url}, request/response # ${k}<br />`;
            res.end(s);
};

let server = http.createServer(http_handler);

server.on('request', (req, res)=>{        // после  http_handler
    console.log('request: # ', k);
});


server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

```
D:\PSCA\Lec06>node 06-05.js
server.listen(3000)
request url: /, #  1
request: #  1
request url: /favicon.ico, #  2
request: #  2
request url: /, #  3
request: #  3
request url: /favicon.ico, #  4
request: #  4
request url: /, #  5
request: #  5
request url: /favicon.ico, #  6
request: #  6
```

6. **HTTP-сервер:** простейший сервер

```
//-------------- 06-06.js -----------------------------------

var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let s = '';

let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать заголовок
            res.write('<h2>Http-сервер</h2>');                                    // отправить порцию
            s += `url = ${req.url}, request/response # ${k}<br />`;
            res.end(s);
};

let server = http.createServer();

server.on('request',http_handler);

server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

7. **HTTP-сервер:** простейший сервер **server.on('conection')**

```
//-------------- 06-07.js -----------------------------------

var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let c = 0;
let s = '';

let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать заголовок
            res.write('<h2>Http-сервер</h2>');                                    // отправить порцию
            s += `url = ${req.url}, request/response # ${c} - ${k}<br />`;
            res.end(s);
};

let server = http.createServer();

server.keepAliveTimeout = 10000;              // время сохранения соединения (connection), умочание  = 5000;
server.on('connection', (socket)=>{           // устанавливается новое соединение
            console.log(`connection: server.keepAliveTimeout = ${server.keepAliveTimeout} `, ++c);
            s += `<h2>connection: # ${c}</h2>`;
});

server.on('request',http_handler);

server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

```
D:\PSCA\Lec06>node 06-07.js
server.listen(3000)
connection: server.keepAliveTimeout = 10000  1
connection: server.keepAliveTimeout = 10000  2
request url: /, #  1
request url: /favicon.ico, #  2
request url: /, #  3
request url: /favicon.ico, #  4
request url: /, #  5
request url: /favicon.ico, #  6
request url: /, #  7
request url: /favicon.ico, #  8
request url: /, #  9
request url: /favicon.ico, #  10
request url: /, #  11
request url: /favicon.ico, #  12
connection: server.keepAliveTimeout = 10000  3
request url: /, #  13
request url: /favicon.ico, #  14
request url: /, #  15
request url: /favicon.ico, #  16
request url: /, #  17
request url: /favicon.ico, #  18
request url: /, #  19
request url: /favicon.ico, #  20
request url: /, #  21
request url: /favicon.ico, #  22
request url: /, #  23
request url: /favicon.ico, #  24
request url: /, #  25
request url: /favicon.ico, #  26
```

localhost:3000

localhost:3000

Http-сервер

connection: # 1

connection: # 2

url = /, request/response # 2 - 1
url = /favicon.ico, request/response # 2 - 2
url = /, request/response # 2 - 3
url = /favicon.ico, request/response # 2 - 4
url = /, request/response # 2 - 5
url = /favicon.ico, request/response # 2 - 6
url = /, request/response # 2 - 7
url = /favicon.ico, request/response # 2 - 8
url = /, request/response # 2 - 9
url = /favicon.ico, request/response # 2 - 10
url = /, request/response # 2 - 11
url = /favicon.ico, request/response # 2 - 12

connection: # 3

url = /, request/response # 3 - 13
url = /favicon.ico, request/response # 3 - 14
url = /, request/response # 3 - 15
url = /favicon.ico, request/response # 3 - 16
url = /, request/response # 3 - 17

8. **HTTP-сервер:** простейший сервер **server.close()**

```
//--------------- 06-07.js -----------------------------------

var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let c = 0;
let s = '';

let server = http.createServer();

let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать заголовок
            res.write('<h2>Http-сервер</h2>');                                    // отправить порцию
            s += `url = ${req.url}, request/response # ${c} - ${k}<br />`;
            res.end(s);
            if (req.url == '/close') {server.close(()=>console.log('server.close'));}
};



server.keepAliveTimeout = 10000;              // время сохранения соединения (connection), умочание  = 5000;
server.on('connection', (socket)=>{           // устанавливается новое соединение
            console.log(`connection: server.keepAliveTimeout = ${server.keepAliveTimeout} `, ++c);
            s += `<h2>connection: # ${c}</h2>`;
});

server.on('request',http_handler);

server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

localhost:3000/close

localhost:3000/close

Приложения Космос ТВ

Http-сервер

connection: # 1

url = /, request/response # 1 - 1

connection: # 2

url = /favicon.ico, request/response # 2 - 2
url = /, request/response # 2 - 3
url = /favicon.ico, request/response # 2 - 4
url = /, request/response # 2 - 5
url = /favicon.ico, request/response # 2 - 6
url = /, request/response # 2 - 7
url = /favicon.ico, request/response # 2 - 8

connection: # 3

connection: # 4

url = /, request/response # 4 - 9
url = /favicon.ico, request/response # 4 - 10

connection: # 5

connection: # 6

url = /close, request/response # 6 - 11

```
D:\PSCA\Lec06>node 06-07.js
server.listen(3000)
connection: server.keepAliveTimeout = 10000  1
request url: /, #  1
connection: server.keepAliveTimeout = 10000  2
request url: /favicon.ico, #  2
request url: /, #  3
request url: /favicon.ico, #  4
request url: /, #  5
request url: /favicon.ico, #  6
request url: /, #  7
request url: /favicon.ico, #  8
connection: server.keepAliveTimeout = 10000  3
connection: server.keepAliveTimeout = 10000  4
request url: /, #  9
request url: /favicon.ico, #  10
connection: server.keepAliveTimeout = 10000  5
connection: server.keepAliveTimeout = 10000  6
request url: /close, #  11
request url: /favicon.ico, #  12
server.close

D:\PSCA\Lec06>
```

9. **HTTP-сервер:** простейший сервер **server.on('close')**

```
let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать заголовок
            res.write('<h2>Http-сервер</h2>');                                    // отправить порцию
            s += `url = ${req.url}, request/response # ${c} - ${k}<br />`;
            res.end(s);
            if (req.url == '/close') {server.close(()=>console.log('server.close'));}
};



server.keepAliveTimeout = 10000;              // время сохранения соединения (connection), умочание  = 5000;
server.on('connection', (socket)=>{           // устанавливается новое соединение
            console.log(`connection: server.keepAliveTimeout = ${server.keepAliveTimeout} `, ++c);
            s += `<h2>connection: # ${c}</h2>`;
});

server.on('request',http_handler);

server.on('close', ()=>{console.log('server.on.close');});

server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

```
D:\PSCA\Lec06>node 06-08.js
server.listen(3000)
connection: server.keepAliveTimeout = 10000  1
request url: /, #  1
connection: server.keepAliveTimeout = 10000  2
request url: /favicon.ico, #  2
request url: /, #  3
request url: /favicon.ico, #  4
request url: /, #  5
request url: /favicon.ico, #  6
request url: /, #  7
request url: /favicon.ico, #  8
connection: server.keepAliveTimeout = 10000  3
request url: /, #  9
request url: /favicon.ico, #  10
request url: /, #  11
request url: /favicon.ico, #  12
connection: server.keepAliveTimeout = 10000  4
connection: server.keepAliveTimeout = 10000  5
request url: /close, #  13
request url: /favicon.ico, #  14
server.on.close
server.close
```

10. **HTTP-сервер:** простейший сервер **server.on('timeout')**

```
//--------------- 06-09.js -----------------------------------

var http   =  require('http');    // низкоуровневый http-сервер

let k = 0;
let c = 0;
let s = '';

let server = http.createServer();

let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать заголовок
            res.write('<h2>Http-сервер</h2>');                                    // отправить порцию
            s += `url = ${req.url}, request/response # ${c} - ${k}<br />`;
            res.end(s);
            if (req.url == '/close') { server.close(()=>console.log('server.close'));}
};

server.keepAliveTimeout = 10000;              // время сохранения соединения (connection), умолчание  = 5000;
server.on('connection', (socket)=>{           // устанавливается новое соединение
            console.log(`connection: server.keepAliveTimeout = ${server.keepAliveTimeout} `, ++c);
            s += `<h2>connection: # ${c}</h2>`;
});

server.on('request',http_handler);

server.timeout = 10000;                       // сообщить о бездействии > 10000, умолчание  = 120000;
server.on('timeout', (socket)=>{console.log('timeout:', server.timeout);});  // server.close() не работает


server.on('close', ()=>{console.log('server.on.close');});

server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
```

11. **HTTP-сервер:** простейший сервер, **socket properties**

```
server.keepAliveTimeout = 10000;              // время сохранения соединения (connection), умолчание  = 5000;
server.on('connection', (socket)=>{           // устанавливается новое соединение
            console.log(`connection: server.keepAliveTimeout = ${server.keepAliveTimeout} `, ++c);
            s += `<h2>connection: # ${c}</h2>`;
            console.log('socket.localAddress = ', socket.localAddress);
            console.log('socket.llocalPort = ',   socket.localPort);
            console.log('socket.remoteAddress = ', socket.remoteAddress);
            console.log('socket.remoteFamily = ', socket.remoteFamily);
            console.log('socket.remotePort = ', socket.remotePort);
            console.log('socket.bytesWritten = ', socket.bytesWritten);
});
```

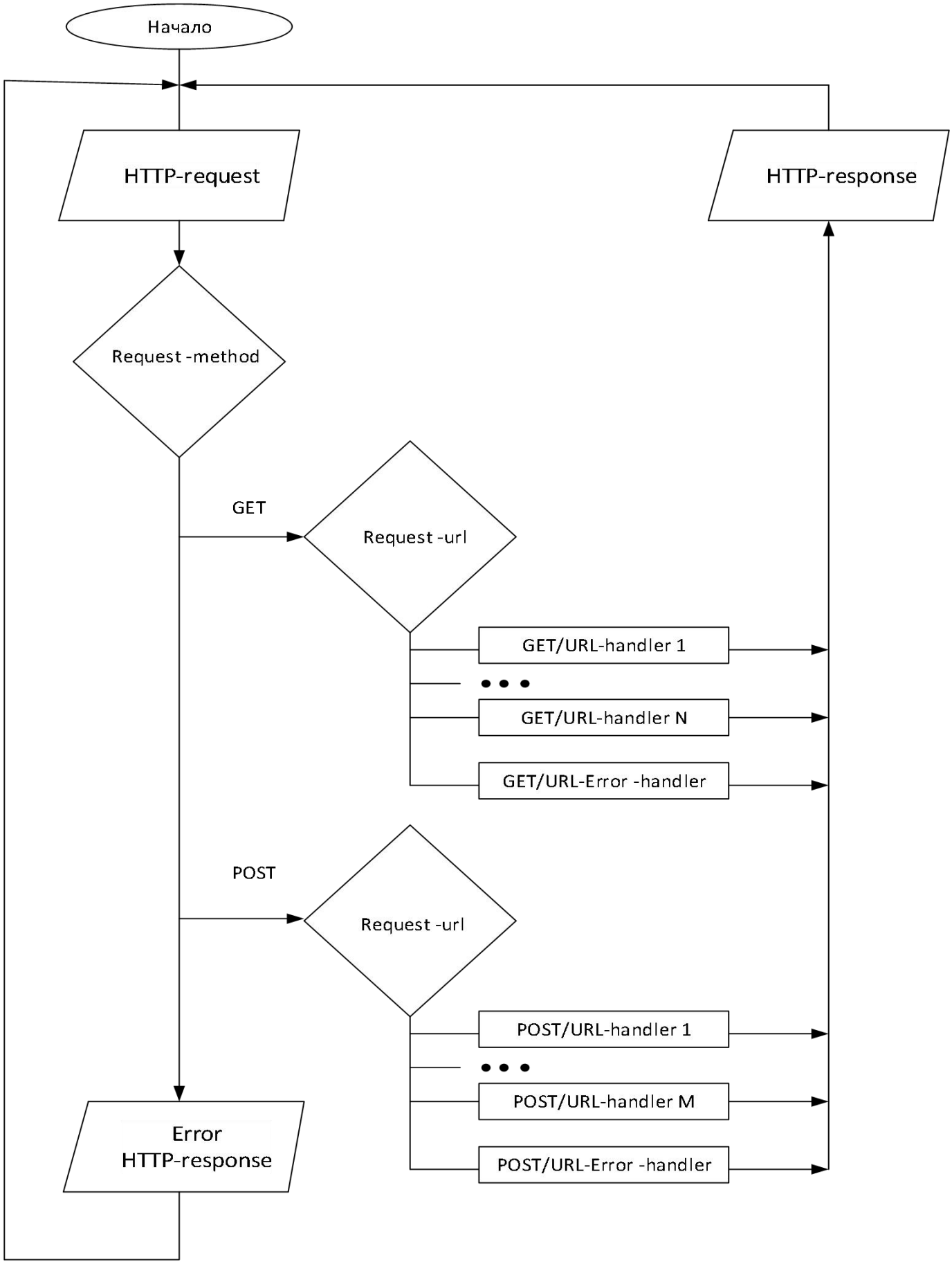
```
D:\PSCA\Lec06>node 06-10.js
server.listen(3000)
connection: server.keepAliveTimeout = 10000  1
socket.localAddress =  ::1
socket.llocalPort =  3000
socket.remoteAddress =  ::1
socket.remoteFamily =  IPv6
socket.remotePort =  64222
socket.bytesWritten =  0
```

12. **HTTP-сервер:** простейший сервер **request.on**

```
let http_handler = (req, res)=>{
            console.log(`request url: ${req.url}, # `, ++k);
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});     //записать заголовок

            let buf='';
            req.on('data', (data)=>{console.log('request.on(data) =', data.length); buf += data;}); // получить фрагментами
            req.on('end',  ()=>{console.log('request.on(end) =', buf.length)})                     // все данные пришли

            res.write('<h2>Http-сервер</h2>');                                        // отправить порцию
            s += `url = ${req.url}, request/response # ${k}<br />`;
            res.end(s);

};
```

```
request url: /, #  11
request.on(data) = 17797
request.on(end) = 17797
request url: /, #  12
request.on(data) = 7826
request.on(data) = 9971
request.on(end) = 17797
request url: /, #  13
request.on(data) = 7795
request.on(data) = 10002
request.on(end) = 17797
request url: /, #  14
request.on(data) = 17797
request.on(end) = 17797
```

13. **HTTP-сервер:** простейший сервер, типичный цикл работы



14. **HTTP-сервер:** простейший сервер, **method**

```
//-------------- 06-12.js -------------------------------------

var http   =  require('http');    // низкоуровневый http-сервер

let debug_handler  = (req, res)=>{
    console.log(req.method, req.url);
    res.writeHead(200, {'Content-Type': 'application/json; charset=utf-8'});
    res.end(`{"${req.method}":"${req.url}}"`);
}

let GET_handler    = (req,res)=>{ debug_handler(req, res)};
let POST_handler   = (req,res)=>{ debug_handler(req, res)};
let PUT_handler    = (req,res)=>{ debug_handler(req, res)};
let DELETE_handler = (req,res)=>{ debug_handler(req, res)};
let OTHER_handler  = (req,res)=>{ debug_handler(req, res)};

let http_handler = (req, res)=>{

        switch (req.method){
            case 'GET' :    GET_handler(req,res);    break;
            case 'POST':    POST_handler(req,res);   break;
            case 'PUT':     PUT_handler(req,res);    break;
            case 'DELETE':  DELETE_handler(req,res); break;
            default:        OTHER_handler(req,res);  break;
        }
};

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
      .on('request', http_handler);
```

```
D:\PSCA\Lec06>node 06-12.js
server.listen(3000)
GET /24
POST /vvv/xxxx
PUT /vvv/xxxx
DELETE /vvv/xxxx/22
PATCH /vvv/xxxx/22
```

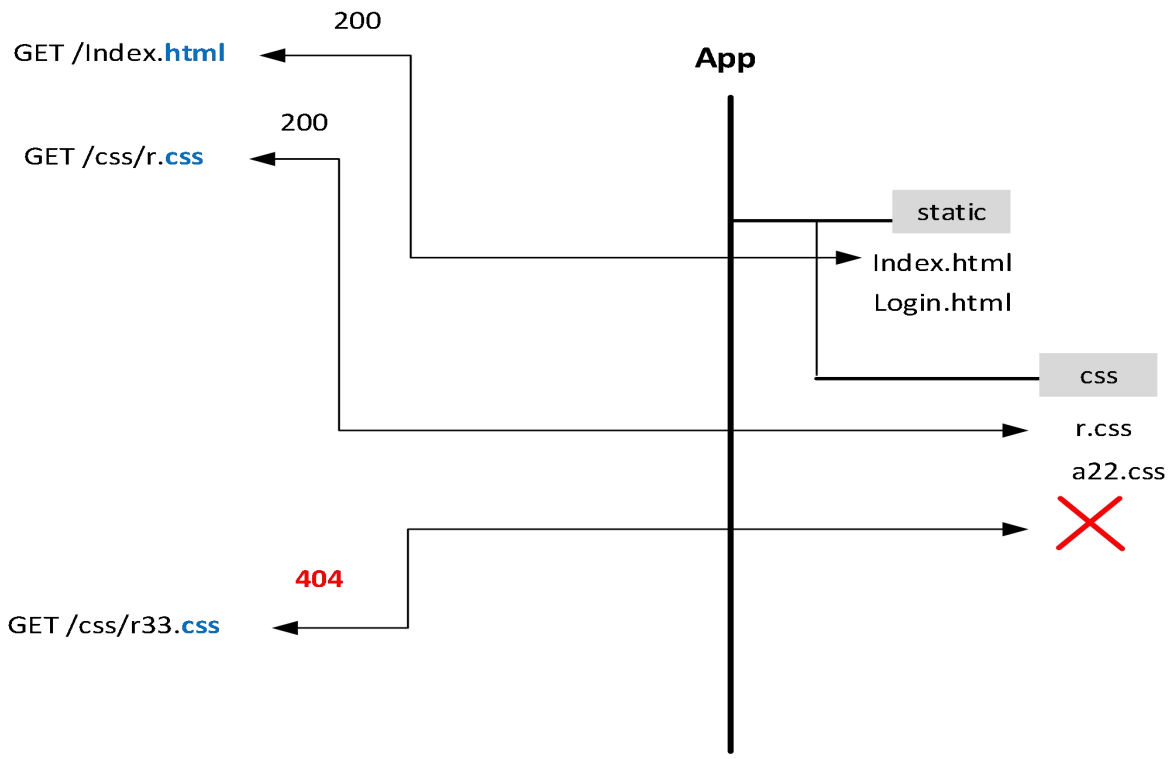
15. **HTTP-сервер:** простейший сервер, url, 404

```
let HTTP404        = (req,res)=>{
    console.log(`${req.method}: ${req.url},  HTTP status 404`);
    res.writeHead(404, {'Content-Type': 'application/json; charset=utf-8'});
    res.end(`{"error":"${req.method}: ${req.url},  HTTP status 404"}`);
};

let GET_handler  = (req,res)=>{ // обработчик get-запросов

    switch (req.url){
        case '/' :              debug_handler(req, res);  break;
        case '/index.html':     debug_handler(req, res);  break;;
        case '/site.css':       debug_handler(req, res);  break;
        case '/calc':           debug_handler(req, res);  break;
        default:                HTTP404(req,res);         break;
    }
};

let http_handler = (req, res)=>{

            switch (req.method){
                case 'GET' :    GET_handler(req,res);     break;
                case 'POST':    POST_handler(req,res);    break;
                case 'PUT':     PUT_handler(req,res);     break;
                case 'DELETE':  DELETE_handler(req,res);  break;
                default:        HTTP404(req,res);         break;
            }
};

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
      .on('request', http_handler);
```

16. **HTTP-сервер:** простейший сервер, request, response

```
var http   =  require('http');    // низкоуровневый http-сервер

                    (parameter) req: any
let http_handler = (req, res)=>{

  console.log('request.url = ', req.url);                    // url
  console.log('request.method = ', req.method);              // HTTP-метод
  console.log('request.httpVersion = ', req.httpVersion);    // HTTP-метод
  // console.log('request.headers = ', req.headers);         // HTTP-заголовки
  for(key in req.headers) console.log(`request header  ${key}: ${ req.headers[key]}`);
  // данные из тела, только через  req.on('data') и req.on('end') и

  //-------------------------------------------------------------------------------
  //
  res.statusCode = 400;                                  // один из способов задать кода статуса
  res.statusMessage = 'Call +375 327 43 76';             // ссообщение для статуса
  res.setHeader('X-author','IS&T, BSTU, smw@belstu.by');  // добавить один заголовок
  res.writeHead(400, {'Content-Type': 'application/json; charset=utf-8'},  // задать кода статуса  добавить один заголовок
                {'Cache-Control':' no-cache'} );                           // и   добавить один  или несколько заголовков

  res.write('{"1" : "1ая порция",');          // писать в тело ответа
  res.write('"2" : "2ая порция",');           // писать в тело ответа
  res.end('"3" : "последняя порция"}');       // писать последнюю порцию данных в тело ответа

};

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
      .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
      .on('request', http_handler);
```

17. **HTTP-сервер:** простейший сервер, статические ресурсы



18. **HTTP-сервер:** простейший сервер, статические ресурсы

```
let http  =  require('http');
let fs    =  require('fs');

let isStatic = (ext, fn)=>{  let reg =  new RegExp(`^\/.+\.${ext}$`); return reg.test(fn);}
let pathStatic = (fn)=>{return  `./static${fn}`; }
let writeHTTP404 = (res)=>{
            res.statusCode = 404;
            res.statusMessage = 'Resourse not found';
            res.end("Resourse not found");
}
let pipeFile = (req, res, headers)=>{
    res.writeHead(200, headers);
    fs.createReadStream(pathStatic(req.url)).pipe(res);
}
let sendFile = (req, res, headers)=>{
    fs.access(pathStatic(req.url), fs.constants.R_OK, err => {
        if(err) writeHTTP404(res);
        else pipeFile(req, res, headers);
    });
}
let http_handler = (req, res)=>{

   if      (isStatic('html', req.url)) sendFile(req, res,  {'Content-Type': 'text/html; charset=utf-8'});
   else if (isStatic('css',  req.url)) sendFile(req, res,  {'Content-Type': 'text/css; charset=utf-8'}) ;
   else if (isStatic('js',   req.url)) sendFile(req, res,  {'Content-Type': 'text/css; charset=utf-8'}) ;
   else writeHTTP404(res);
};

let server = http.createServer();
    server.listen(3000, (v)=>{console.log('server.listen(3000)')})
          .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
          .on('request', http_handler);
```

19. **HTTP-сервер:** простейший сервер, статические ресурсы, параметризируемый модуль.

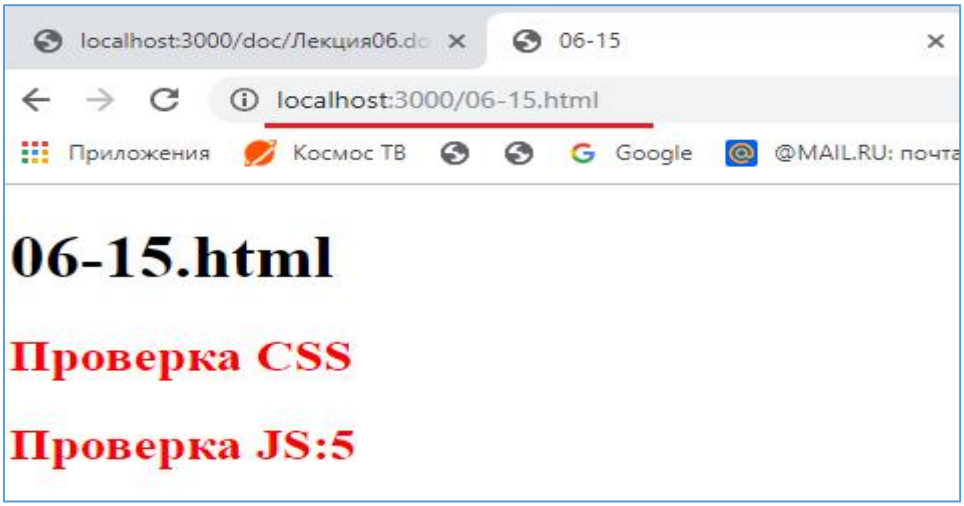
```
let http    =  require('http');
let stat    =  require('./m06-16a')('./static');

let http_handler = (req, res)=>{
   if      (stat.isStatic('html', req.url)) stat.sendFile(req, res,  {'Content-Type': 'text/html; charset=utf-8'});
   else if (stat.isStatic('css',  req.url)) stat.sendFile(req, res,  {'Content-Type': 'text/css; charset=utf-8'}) ;
   else if (stat.isStatic('js',   req.url)) stat.sendFile(req, res,  {'Content-Type': 'text/javascript; charset=utf-8'}) ;
   else if (stat.isStatic('docx', req.url)) stat.sendFile(req, res,  {'Content-Type': 'application/msword'}) ;
   else stat.writeHTTP404(res);
};

let server = http.createServer();
    server.listen(3000, (v)=>{console.log('server.listen(3000)')})
          .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
          .on('request', http_handler);
```

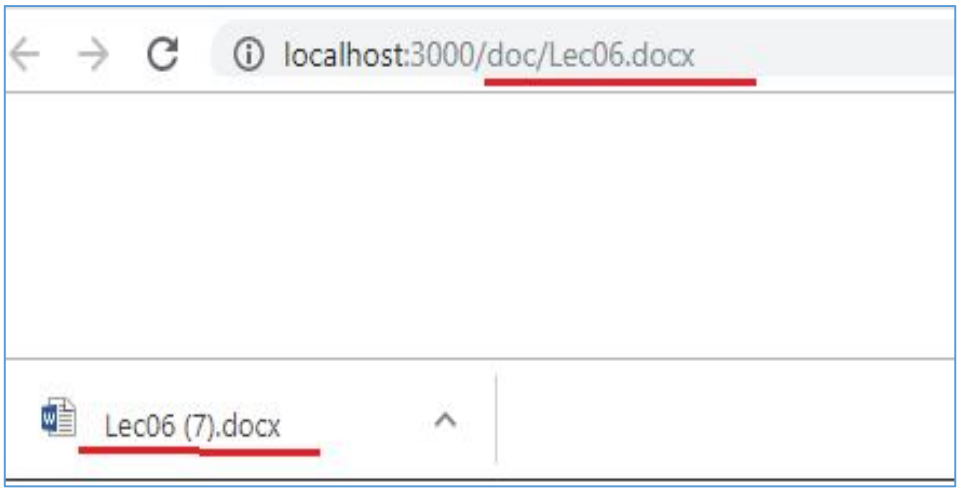
```js
function Stat( sfn = './static'){
    this.STATIC_FOLDER = sfn;
    let pathStatic = (fn)=>{return  `${this.STATIC_FOLDER}${fn}`; }
    this.writeHTTP404 = (res)=>{
        res.statusCode = 404;
        res.statusMessage = 'Resourse not found';
        res.end("Resourse not found");
    }
    let fs    =  require('fs');
    let pipeFile = (req, res, headers)=>{
        res.writeHead(200, headers);
        fs.createReadStream(pathStatic(req.url)).pipe(res);
    }
    this.isStatic = (ext, fn)=>{  let reg =  new RegExp(`^\/.+\.${ext}$`); return reg.test(fn);}
    this.sendFile = (req, res, headers)=>{
        fs.access(pathStatic(req.url), fs.constants.R_OK, err => {
            if(err) this.writeHTTP404(res);
            else pipeFile(req, res, headers);
        });
    }
}

module.exports =  (parm)=>{ return new Stat(parm);}
```

PSCA › Lec06 › static ›

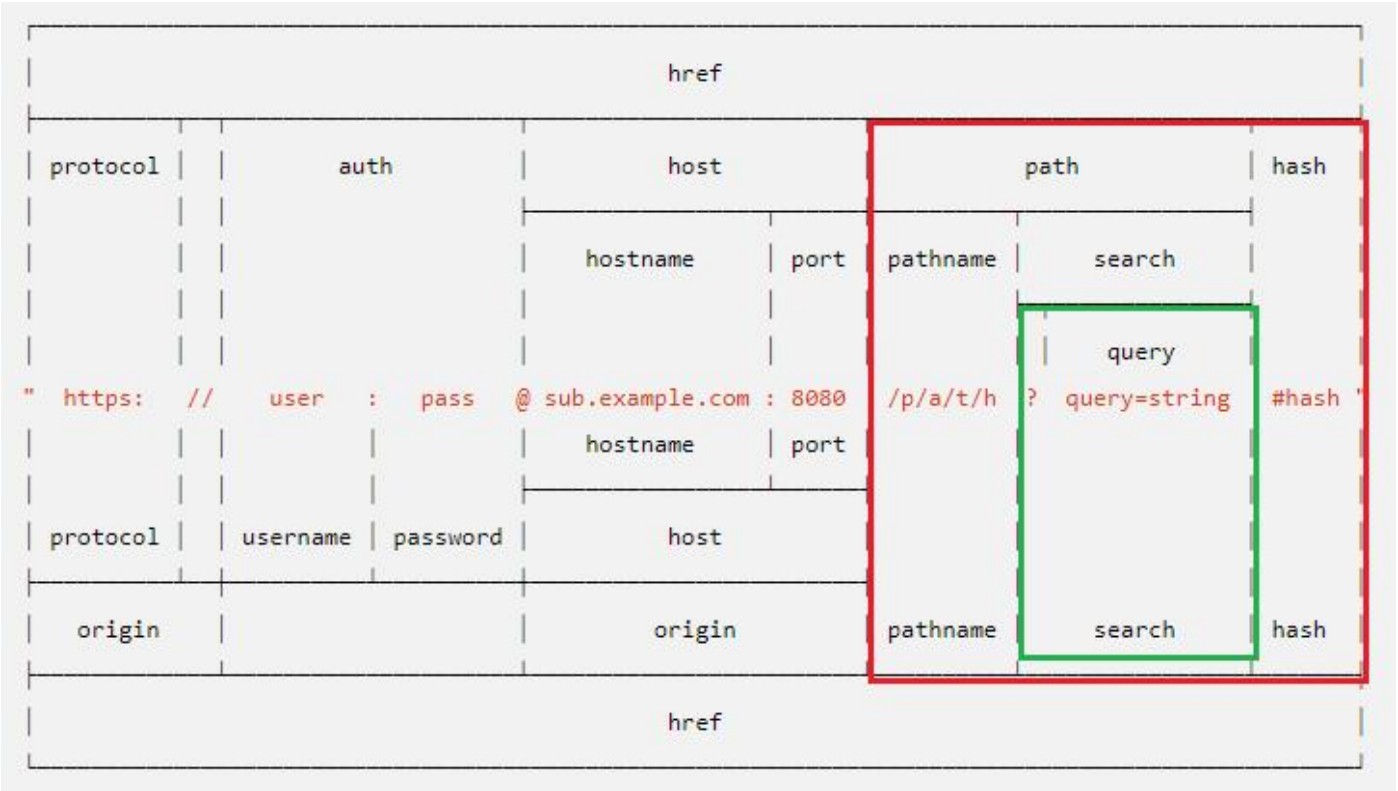
Имя	Дата изменения	Тип	Раз
css	23.08.2019 10:41	Папка с файлами	
doc	23.08.2019 13:39	Папка с файлами	
js	23.08.2019 11:13	Папка с файлами	
06-15.html	23.08.2019 11:25	Chrome HTML Do...	

```html
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>06-15</title>
    <link rel="stylesheet" href="css/06-15.css">
    <script src="js/06-15.js"></script>
</head>
<body>
  <h1>06-15.html</h1>
  <h2 class="danger">Проверка CSS</h2>
  <h2 id = "result" class="danger" >Проверка JS:</h2>
  <script>
    result.innerHTML += sum(3,2);
  </script>
</body>
</html>
```





20. **HTTP-сервер:** простейший сервер, обработка параметров GET-запроса.



21. **HTTP-сервер:** простейший сервер, query-параметры, GET-запросы.

```
let http    =  require('http');
let url     =  require('url');

let handler = (req, res)=>{
   if (req.method = 'GET'){
            let p = url.parse(req.url,true);
            let result ='';
            let q =  url.parse(req.url,true).query;
            if (!(p.pathname == '/favicon.ico')){
                result = `href: ${p.href}<br/>` +
                         `path: ${p.path}<br/>` +
                         `pathname: ${p.pathname}<br/>` +
                         `search: ${p.search}<br/>`;
                for(key in q) { result+= `${key} = ${q[key]}<br/>`;}
            }
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
            res.write('<h1>GET-параметры</h1>');
            res.end(result);
    }
    else{
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.end('for other http-methods not so');
    }
}

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
        .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
        .on('request', handler)
```

GET-параметры

href: /hhh/?k=3&s=kkkk&j=iii&p1=3&p2=t
path: /hhh/?k=3&s=kkkk&j=iii&p1=3&p2=t
pathname: /hhh/
search: ?k=3&s=kkkk&j=iii&p1=3&p2=t
k = 3
s = kkkk
j = iii
p1 = 3
p2 = t

22. **HTTP-сервер:** простейший сервер, url-параметры,GET-запросы.

```
let http    =  require('http');
let url     =  require('url');

let handler = (req, res)=>{
   if (req.method = 'GET'){
            let p = url.parse(req.url,true);
            let result ='';
            let q =  url.parse(req.url,true).query;
            if (!(p.pathname == '/favicon.ico')){
                result = `pathname: ${p.pathname}<br/>`;
                p.pathname.split('/').forEach(e => {result+=  `${e}<br/>`});
            }
            console.log(p.pathname.split('/'));
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
            res.write('<h1>URL-параметры</h1>');
            res.end(result);
    }
    else{
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.end('for other http-methods not so');
    }
}

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
        .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
        .on('request', handler)
```

localhost:3000/БГТУ/ИТ/ПОИТ/

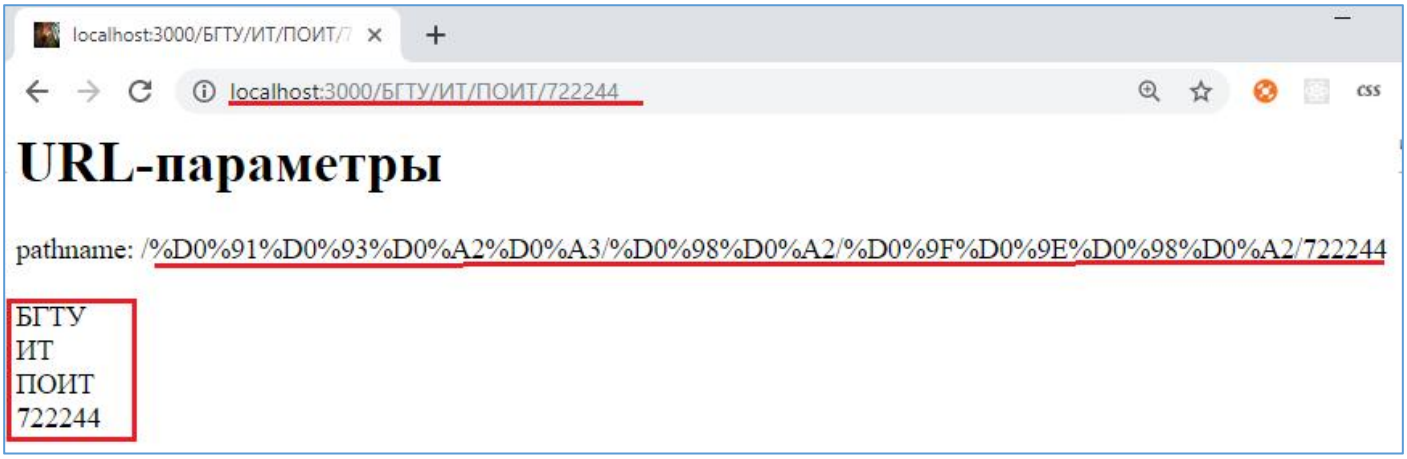
localhost:3000/БГТУ/ИТ/ПОИТ/722244

URL-параметры

pathname: /%D0%91%D0%93%D0%A2%D0%A3/%D0%98%D0%A2/%D0%9F%D0%9E%D0%98%D0%A2/722244

%D0%91%D0%93%D0%A2%D0%A3
%D0%98%D0%A2
%D0%9F%D0%9E%D0%98%D0%A2
722244

```
if (req.method = 'GET'){
        let p = url.parse(req.url,true);
        let result ='';
        let q =  url.parse(req.url,true).query;
        if (!(p.pathname == '/favicon.ico')){
            result = `pathname: ${p.pathname}<br/>`;
            decodeURI(p.pathname).split('/').forEach(e => {result+=  `${e}<br/>`});
        }
        console.log(p.pathname.split('/'));
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.write('<h1>URL-параметры</h1>');
        res.end(result);
}
```



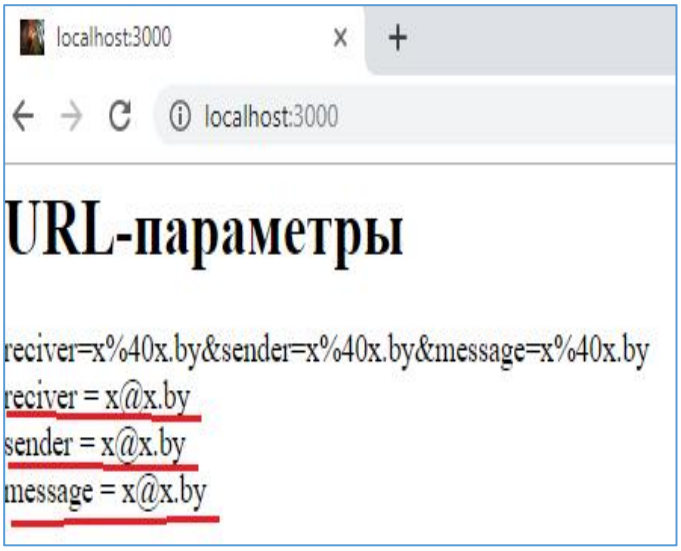
23. **HTTP-сервер:** обработка GET-параметров

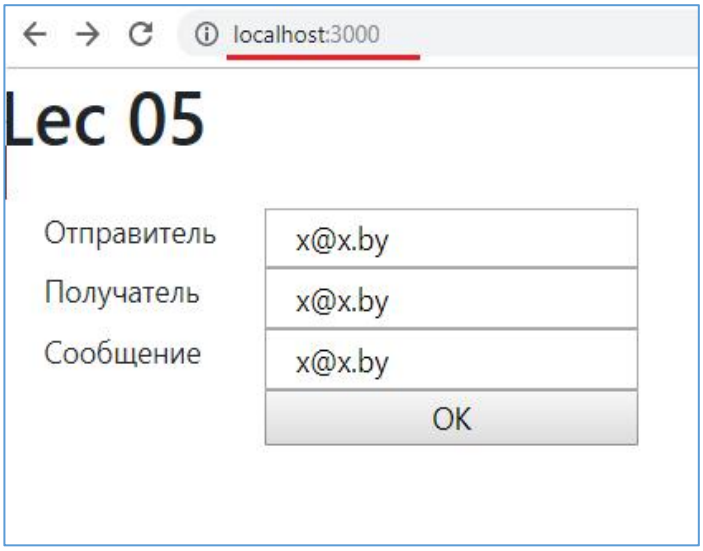
Модуль	Функции, классы
url	Parse/format, classes: URL, URLSearchParams
querystring	parse/stringify, escape/unescspe

24. **HTTP-сервер:** простейший сервер, POST-параметры

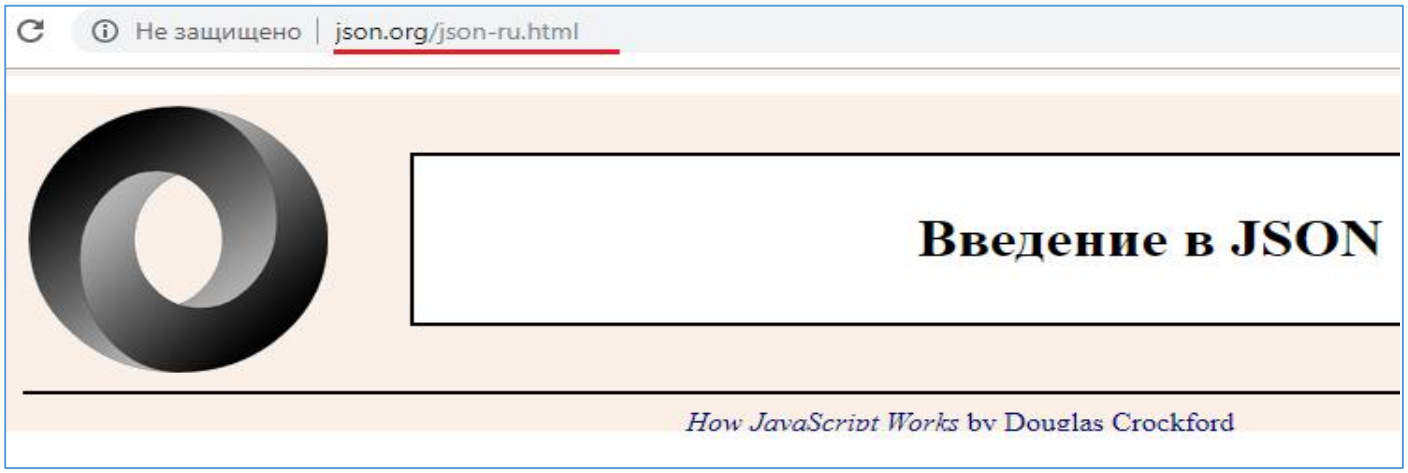
```
let http    =  require('http');
let fs      =  require('fs');
let qs      =  require('querystring');

let handler = (req, res)=>{
    if (req.method == 'GET'){
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.end(fs.readFileSync('./07-03.html'));
    }
    else if (req.method == 'POST'){
            let result = '';
            req.on('data', (data)=>{result+=data;})
            req.on('end',  ()=>{
                result += '<br/>';
                let o = qs.parse(result)
                for (let key in o) { result += `${key} = ${o[key]}<br />`}
                res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                res.write('<h1>URL-параметры</h1>');
                res.end(result);
            });
    }
    else{
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.end('for other http-methods not so');
    }
}

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
        .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
        .on('request', handler)
```

```
<body>
<h1>Lec 05</h1>
<div style="margin: 20px; width: 800px;  padding: 5px;">
    <form method="POST" action="/" >
        <div class ="row">
            <label class="col-2">Отправитель</label> <input class="col-3" name ="reciver"  placeholder
        </div>
        <div class ="row">
            <label class="col-2">Получатель</label>  <input class="col-3" name ="sender"   placeholder
        </div>
        <div class ="row">
            <label class="col-2">Сообщение</label>   <input class="col-3" name = "message" placeholder
        </div>
        <div class ="row">
            <input type="submit" class="col-3 offset-2" value="OK"/>
        </div>
      </form>
</div>
```





25. **HTTP-сервер:** простейший сервер, JSON-формат
26. **JSON:** JavaScript Object Notation, текстовый формат передачи данных, автор: Дуглас Крокфорд,




```
let obj1 = {
        x: 1,
        y: 1.0123456789,
        s: 'Строка',
        m:['a','b','c','d'],
        o:{surname:'Иванов', name:'Иван'}
}
console.log('-----------------------------------------------')
console.log('obj1 = ', obj1);
console.log('-----------------------------------------------')
let json_obj1 = JSON.stringify(obj1);
console.log('json_obj1 = ', json_obj1);
console.log('-----------------------------------------------')
let obj2 = JSON.parse(json_obj1);
console.log('obj2 = ', obj2);
```

```
D:\PSCA\Lec07>node 07-04
---------------------------------------------
obj1 =  { x: 1,
  y: 1.0123456789,
  s: 'Строка',
  m: [ 'a', 'b', 'c', 'd' ],
  o: { surname: 'Иванов', name: 'Иван' } }
---------------------------------------------
json_obj1 =  {"x":1,"y":1.0123456789,"s":"Строка","m":["a","b","c","d"],"o":{"surname":"Иванов","name":"Иван"}}
---------------------------------------------
obj2 =  { x: 1,
  y: 1.0123456789,
  s: 'Строка',
  m: [ 'a', 'b', 'c', 'd' ],
  o: { surname: 'Иванов', name: 'Иван' } }
```

27. **JSON: require:** часто применяется для конфигурационных файлов

```
f07-05.json > ...
{
    "__comment": "Так можно сделать комментарий, другого способа нет",
    "x": 1,
    "y": 1.0123456789,
    "s": "Строка",
    "m":["a","b","c","d"],
    "o":{"surname":"Иванов", "name":"Иван"}
}
```

```
let obj = require('./f07-05.json');
console.log('obj = ', obj);
```

```
D:\PSCA\Lec07>node 07-05
obj =  { __comment: 'Так можно сделать комментарий, другого саособа нет',
  x: 1,
  y: 1.0123456789,
  s: 'Строка',
  m: [ 'a', 'b', 'c', 'd' ],
  o: { surname: 'Иванов', name: 'Иван' } }
```

28. **JSON: MIME:** Multipurpose Internet Mail Extensions, ***application/json*** (RFC 4627), Content-Type, Accept

tools.ietf.org/html/rfc4627

[Docs] [txt|pdf] [draft-crockford...] [Tracker] [Diff1] [Diff2] [Errata]

Obsoleted by: 7159 INFORMATIONAL
Errata Exist

Network Working Group D. Crockford
Request for Comments: 4627 JSON.org
Category: Informational July 2006

The application/json Media Type for JavaScript Object Notation (JSON)

29. **HTTP-сервер:** простейший сервер, JSON-формат

```
let http    =  require('http');
let m0706   =  require('./m07-06');

let handler = (req, res)=>{
    if (req.method == 'POST' &&  m0706.isJsonContentType (req.headers)){
            let result = '';
            req.on('data', (data)=>{result += data;})
            req.on('end',  ()=>{
                try {
                    let obj = JSON.parse(result);
                    console.log(obj);
                    if (m0706.isJsonAccept(req.headers))
                            m0706.write200(res, 'ok json', JSON.stringify(obj));
                    else m0706.write400(res, 'no accept');
                }
                catch (e){m0706.write400(res, 'catch: bad json');}
            })
    } else  m0706.write400(res, 'no json-post');
}

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
        .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
        .on('request', handler)
```

```
const isJson = (headers, header,  mime) => {
    let rc = false;
    let  h = headers[header];
    if (h) rc =  h.indexOf(mime) >= 0;
    return rc;
}
exports.write400 = (res, smess)=>{
    console.log(smess);
    res.writeHead(400, {'Content-Type': 'text/html; charset=utf-8'});
    res.statusMessage = smess;
    res.end();
}
exports.write200 = (res, smess, mess)=>{
    console.log(smess, mess);
    res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
    res.statusMessage = smess;
    res.end(mess);
}
exports.isJsonContentType = (hs) => isJson(hs,'content-type', 'application/json');
exports.isJsonAccept      = (hs) => isJson(hs,'accept',       'application/json');
```

30. **HTTP-сервер:** простейший сервер, XML-формат, MIME: application/xml, text/xml

tools.ietf.org/html/rfc3023

[Docs] [txt|pdf] [draft-murata-xml] [Tracker] [Diff1] [Diff2] [Errata]

```
Obsoleted by: 7303                                    PROPOSED STANDARD
Updated by: 6839                                          Errata Exist
Network Working Group                                        M. Murata
Request for Comments: 3023                IBM Tokyo Research Laboratory
Obsoletes: 2376                                          S. St.Laurent
Updates: 2048                                             simonstl.com
Category: Standards Track                                     D. Kohn
                                                       Skymoon Ventures
                                                           January 2001
```

XML Media Types

```
let parseString = require('xml2js').parseString;  // npm install xml2js
let xmlbuilder  = require('xmlbuilder');          // скачивается в одном пакете с xml2js
```

```
let xmltext = '<?xml version="1.0" encoding="utf-8"?>'+   // не обязательно
            '<students faculty="ИТ" sprciality="ИСиТ" >' +
            '<student  id="7000222"  name="Иванов И.И."    bday="2000-12-02" />'+
            '<student  id="7000223"  name="Петров П.П."    bday="2000-11-28" />'+
            '<student  id="7000228"  name="Казан Н.А." bday="2001-09-11" />'+
            '</students>';
```

```
let obj = null;
```

```
parseString(xmltext, function (err, result) {
                obj = result;
                console.log('---------- parseString ----------------');
                console.log(err);
                console.log('--------------------------');
                console.log('result = ', result);
                console.log('--------------------------');
                result.students.student.map( (e,i)=>{
                    console.log(`id = ${e.$.id}, name = ${e.$.name}, name = ${e.$.bday}`);
                })
});
```

```
console.log('-------- xmlbuilder ------------------');
let xml2 = xmlbuilder.create(obj,
    {version: '1.0', encoding: 'UTF-8', standalone: true}
).end({pretty: true, standalone: true});
console.log(xml2);
```

```
D:\PSCA\Lec07>node 07-07
---------- parseString ----------------
null
--------------------------
result =  { students:
   { '$': { faculty: 'ИТ', sprciality: 'ИСиТ' },
     student: [ [Object], [Object], [Object] ] } }
--------------------------
id = 7000222, name = Иванов И.И., name = 2000-12-02
id = 7000223, name = Петров П.П., name = 2000-11-28
id = 7000228, name = Казан Н.А., name = 2001-09-11
-------- xmlbuilder ------------------
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students>
  <$>
    <faculty>ИТ</faculty>
    <sprciality>ИСиТ</sprciality>
  </$>
  <student>
    <$>
      <id>7000222</id>
      <name>Иванов И.И.</name>
      <bday>2000-12-02</bday>
    </$>
  </student>
  <student>
    <$>
      <id>7000223</id>
      <name>Петров П.П.</name>
      <bday>2000-11-28</bday>
    </$>
  </student>
  <student>
    <$>
      <id>7000228</id>
      <name>Казан Н.А.</name>
      <bday>2001-09-11</bday>
    </$>
  </student>
</students>
```

```
let xmlbuilder  = require('xmlbuilder');   // скачивается в одном пакете с xml2js

// https://github.com/oozcitak/xmlbuilder-js/wiki

let xmldoc  = xmlbuilder.create('students').att('faculty', 'ИТ').att('speciality','ИСиТ');
    xmldoc.ele('student').att('id', '7000222').att('name','Иванов И.И.').att('bday','2000-12-02')
     .up().ele('student').att('id', '7000223').att('name','Петров П.П.').att('bday','2000-11-29')
            .txt('Прошел собеседование в iTechArt')
     .up().ele('student').att('id', '7000228').att('name','Казан Н.А.') .att('bday','2001-09-11');


let xmldoc1 =  xmlbuilder.create('students').att('faculty', 'ИТ').att('speciality','ИСиТ');
    xmldoc1.ele('student', {id: '7000222', name:'Иванов И.И.', bday:'2000-12-02'});
    xmldoc1.ele('student', {id: '7000223', name:'Петров П.П.', bday:'2000-11-29'})
            .txt('Прошел собеседование в iTechArt');
    xmldoc1.ele('student', {id: '7000223', name:'Казан Н.А.', bday:'2001-09-11'});


console.log(xmldoc.toString({pretty:true}));
console.log(xmldoc1.toString({pretty:true}));
```

```
D:\PSCA\Lec07>node 07-08
<students faculty="ИТ" speciality="ИСиТ">
  <student id="7000222" name="Иванов И.И." bday="2000-12-02"/>
  <student id="7000223" name="Петров П.П." bday="2000-11-29">Прошел собеседование в iTechArt</student>
  <student id="7000228" name="Казан Н.А." bday="2001-09-11"/>
</students>

<students faculty="ИТ" speciality="ИСиТ">
  <student id="7000222" name="Иванов И.И." bday="2000-12-02"/>
  <student id="7000223" name="Петров П.П." bday="2000-11-29">Прошел собеседование в iTechArt</student>
  <student id="7000223" name="Казан Н.А." bday="2001-09-11"/>
</students>
```

```
let http     =  require('http');
let parseString = require('xml2js').parseString;
let xmlbuilder  = require('xmlbuilder');
let m0709    =  require('./m07-09');

let studentscalc = (obj)=>{
    let rc = '<result>parse error</result>';
    try {
        let xmldoc  = xmlbuilder.create('result');
        xmldoc.ele('students').att('faculty', obj.students.$.faculty).att('speciality',obj.students.$.speciality)
                .ele('quantity').att('value', obj.students.student.length);
        rc =  xmldoc.toString({pretty:true});
    }catch(e){console.log(e);}
        return rc
}

let handler = (req, res)=>{
    if (req.method == 'POST' &&  m0709.isXMLContentType (req.headers)){
        if (m0709.isXMLAccept(req.headers)) {
           let xmltxt = '';
            req.on('data', (data)=>{xmltxt += data;})
            req.on('end',  ()=>{
                    parseString(xmltxt, function (err, result) {
                        if (err)  m0709.write400(res, 'xml parse error');
                        else m0709.write200(res, 'ok xml', studentscalc(result));
                    })
            })
        }
        else m0709.write400(res, 'no xml accept');
    } else  m0709.write400(res, 'no xml-post');
}

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
        .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
        .on('request', handler)
```

POST http://localhost:3000 Send

Params Authorization Headers (9) Body Pre-request Script Tests Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL BETA XML (text/xml)

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <students faculty="ИТ" speciality="ИСиТ" >
3 <student  id="7000222"  name="Иванов И.И."  bday="2000-12-02" />
4 <student  id="7000223"  name="Петров П.П."  bday="2000-11-28" />
5 <student  id="7000228"  name="Казан Н.А."   bday="2001-09-11" />
6 </students>
```

Body Cookies Headers (4) Test Results Status: 200 OK Time: 15ms Size: 256 B Save

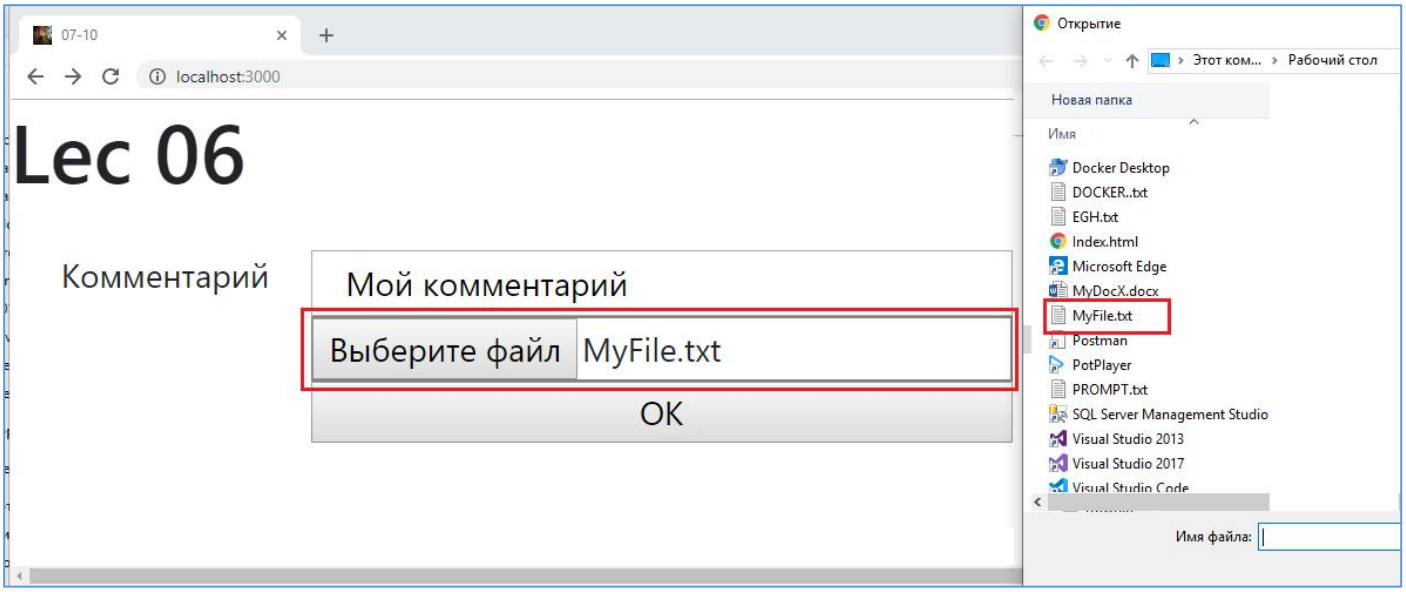
Pretty Raw Preview XML

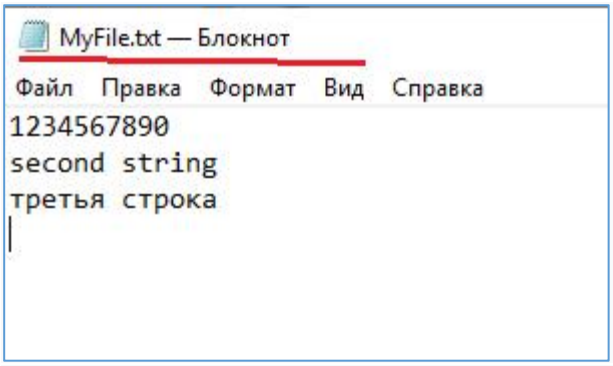
```
1 <result>
2     <students faculty="ИТ" speciality="ИСиТ">
3         <quantity value="3"/>
4     </students>
5 </result>
```

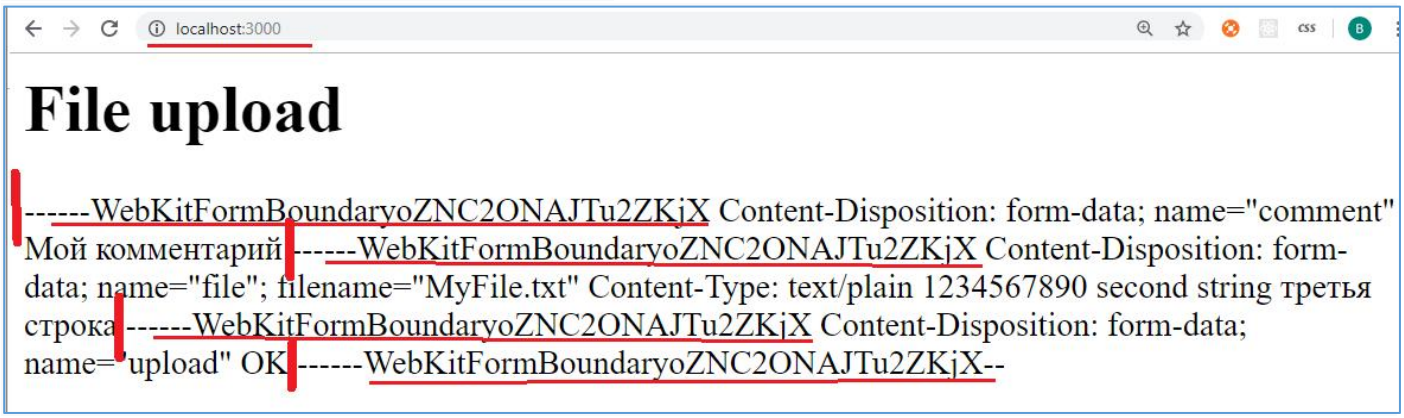
31. **HTTP-сервер:** простейший сервер, file upload

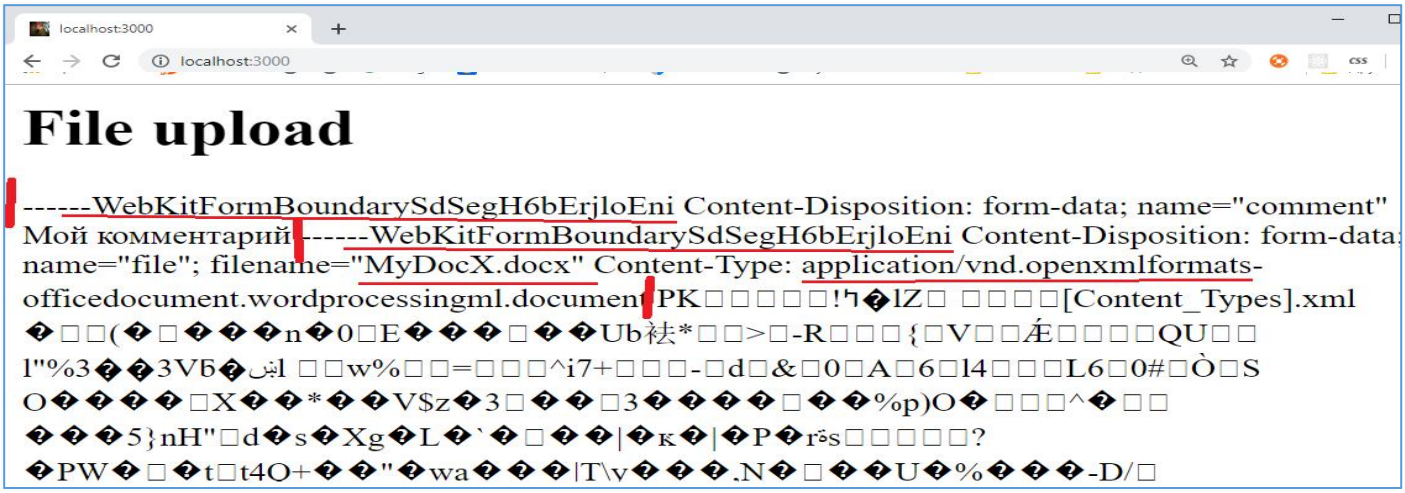
```
let http    =  require('http');
let fs      =  require('fs');

let handler = (req, res)=>{
    if (req.method == 'GET'){
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.end(fs.readFileSync('./07-10.html'));
    }
    else if (req.method == 'POST'){
            let result = '';
            req.on('data', (data)=>{result+=data;})
            req.on('end',  ()=>{
                res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                res.write('<h1>File upload</h1>');
                res.end(result);
            });
    }
    else{
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.end('for other http-methods not so');
    }
}

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
        .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
        .on('request', handler)
```

```
<!DOCTYPE html>
<html >
<head>
    <meta name="viewport" content="width=device-width" charset="utf-8"  />
    <title>07-10</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
<h1>Lec 06</h1>
<div style="margin: 20px; width: 800px;  padding: 5px;">
    <form method="POST" action="/"  enctype="multipart/form-data">
        <div class ="row">
            <label class="col-2">Комментарий</label> <input class="col-5" name ="comment"   type="text" />
        </div>
        <div class ="row">
            <input  class="col-5 offset-2" style="padding: 0px; border: 1px solid gray;"  name ="file"  type="file" />
        </div>
        <div class ="row">
            <input type="submit" class="col-5 offset-2" name="upload"  value="OK"/>
        </div>
    </form>
</div>
```


07-10
localhost:3000
Lec 06
Комментарий
Мой комментарий
Выберите файл
MyFile.txt
OK
Открытие
Этот ком...
Рабочий стол
Новая папка
Имя
Docker Desktop
DOCKER..txt
EGH.txt
Index.html
Microsoft Edge
MyDocX.docx
MyFile.txt
Postman
PotPlayer
PROMPT.txt
SQL Server Management Studio
Visual Studio 2013
Visual Studio 2017
Имя файла:


MyFile.txt — Блокнот
Файл Правка Формат Вид Справка
1234567890
second string
третья строка

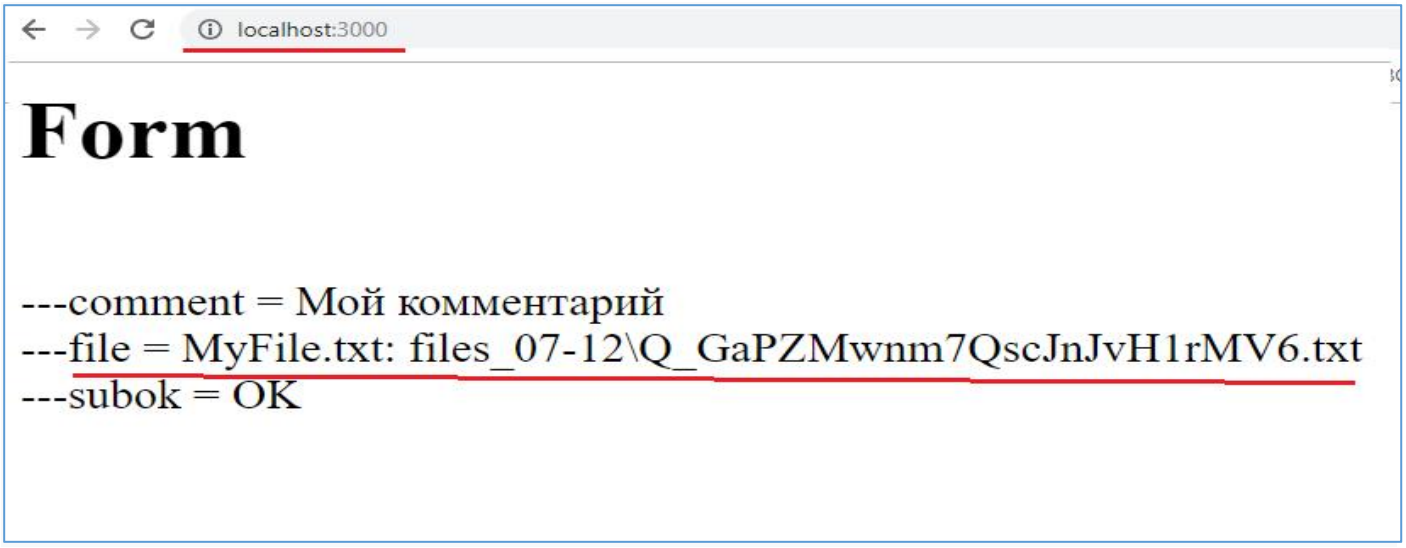

localhost:3000
File upload
------WebKitFormBoundaryoZNC2ONAJTu2ZKjX Content-Disposition: form-data; name="comment" Мой комментарий ------WebKitFormBoundaryoZNC2ONAJTu2ZKjX Content-Disposition: form-data; name="file"; filename="MyFile.txt" Content-Type: text/plain 1234567890 second string третья строка ------WebKitFormBoundaryoZNC2ONAJTu2ZKjX Content-Disposition: form-data; name="upload" OK ------WebKitFormBoundaryoZNC2ONAJTu2ZKjX--

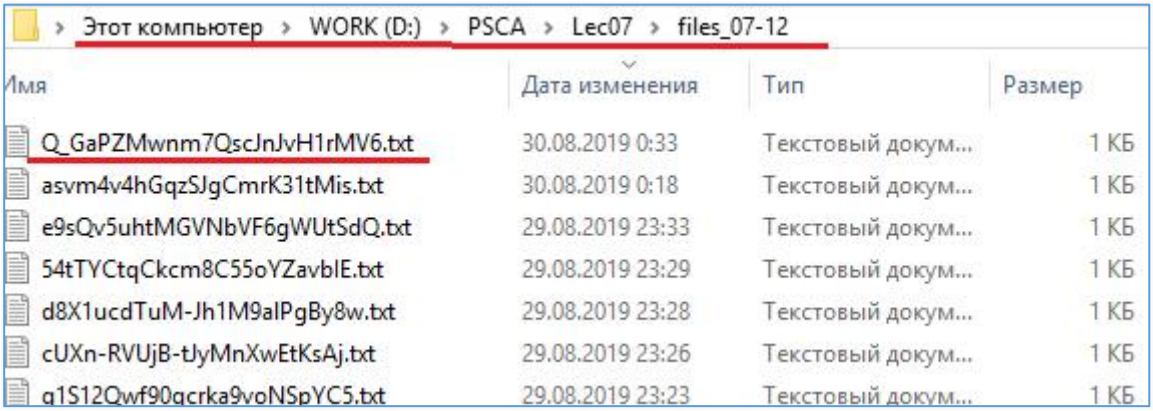

localhost:3000
File upload
------WebKitFormBoundarySdSegH6bErjloEni Content-Disposition: form-data; name="comment" Мой комментарий ------WebKitFormBoundarySdSegH6bErjloEni Content-Disposition: form-data; name="file"; filename="MyDocX.docx" Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document PK□□□□□!ㄱ□lZ□ □□□□[Content_Types].xml

32. **HTTP-сервер:** простейший сервер, file upload, multiparty

```
let http    = require('http');
let fs      = require('fs');
let mp      = require('multiparty');  // npm install multiparty  // https://github.com/pillarjs/multiparty
```

```
let handler = (req, res)=>{
    if (req.method == 'GET'){
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        res.end(fs.readFileSync('./07-12.html'));
    }
    else if (req.method == 'POST'){
            let result = '';
            // req.on('data', (data)=>{result+=data;})  // все внутри  multiparty
            // req.on('end',  ()=>{ });                  // -------------------
            let  form = new mp.Form({uploadDir:'./files_07-12'});
            form.on('field', (name, value)=>{
                console.log('----- field ------------------');
                console.log(name, value);
                result += `<br />---${name} = ${value}`;
            });
            form.on('file', (name, file)=>{
                console.log('----- file -------------------');
                console.log(name, file);
                result += `<br />---${name} = ${file.originalFilename}: ${file.path}`;
            });
            form.on('error', (err)=> {
                console.log('----- err --------------------');
                console.log('err =', err);
                res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                res.write('<h1>Form/Error</h1>');
                res.end();
            });
            form.on('close', ()=> {
                console.log('----- close ------------------');
                res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                res.write('<h1>Form</h1>');
                res.end(result);
            });
            form.parse(req);
    }
}

let server = http.createServer();
server.listen(3000, (v)=>{console.log('server.listen(3000)')})
        .on('error',  (e)=>{console.log('server.listen(3000): error: ', e.code)})
        .on('request', handler)
```






```
http.STATUS_CODES
{ '100': 'Continue',
  '101': 'Switching Protocols',
  '102': 'Processing',
  '200': 'OK',
  '201': 'Created',
  '202': 'Accepted',
  '203': 'Non-Authoritative Information',
  '204': 'No Content',
  '205': 'Reset Content',
  '206': 'Partial Content',
  '207': 'Multi-Status',
  '300': 'Multiple Choices',
  '301': 'Moved Permanently',
  '302': 'Moved Temporarily',
  '303': 'See Other',
  '304': 'Not Modified',
  '305': 'Use Proxy',
  '307': 'Temporary Redirect',
  '400': 'Bad Request',
  '401': 'Unauthorized',
  '402': 'Payment Required',
  '403': 'Forbidden',
  '404': 'Not Found',
  '405': 'Method Not Allowed',
  '406': 'Not Acceptable',
  '407': 'Proxy Authentication Required',
```

```
'408': 'Request Time-out',
'409': 'Conflict',
'410': 'Gone',
'411': 'Length Required',
'412': 'Precondition Failed',
'413': 'Request Entity Too Large',
'414': 'Request-URI Too Large',
'415': 'Unsupported Media Type',
'416': 'Requested Range Not Satisfiable',
'417': 'Expectation Failed',
'418': 'I\'m a teapot',
'422': 'Unprocessable Entity',
'423': 'Locked',
'424': 'Failed Dependency',
'425': 'Unordered Collection',
'426': 'Upgrade Required',
'428': 'Precondition Required',
'429': 'Too Many Requests',
'431': 'Request Header Fields Too Large',
'500': 'Internal Server Error',
'501': 'Not Implemented',
'502': 'Bad Gateway',
'503': 'Service Unavailable',
'504': 'Gateway Time-out',
'505': 'HTTP Version Not Supported',
'506': 'Variant Also Negotiates',
'507': 'Insufficient Storage',
'509': 'Bandwidth Limit Exceeded',
'510': 'Not Extended',
'511': 'Network Authentication Required' }
```