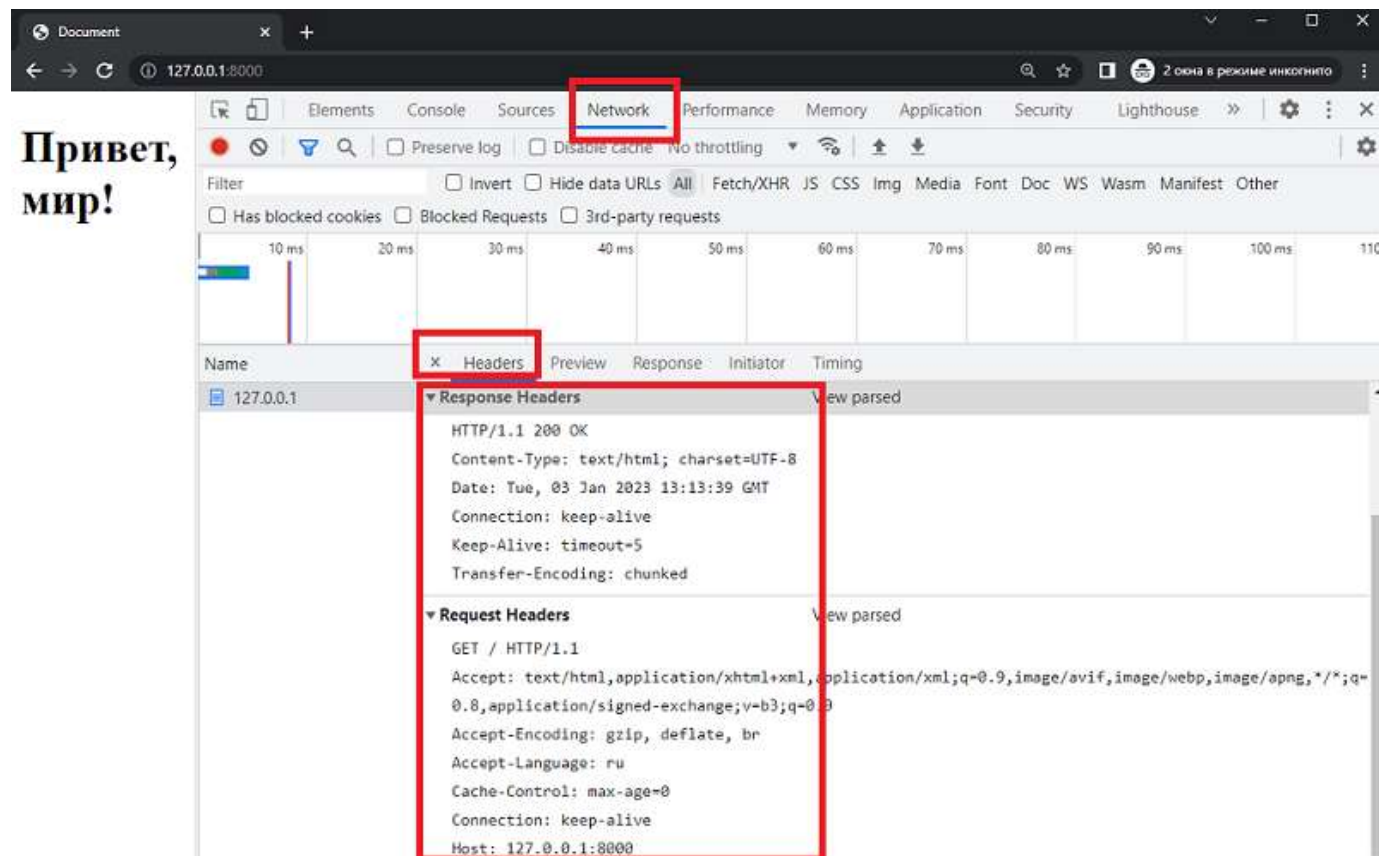


# Лабораторная работа 7: Работа с HTTP-сервером

В этой работе мы создадим и будем работать с встроенным в Nodejs объектом `http` - HTTP-сервером

- Откройте в браузере произвольную страницу в сети, зайдите в панель разработчика, выберите *Network* и обновите страницу - будут видны HTTP-заголовки



- Откройте в консоли папку **07-lab-http-server** и инициализируйте её командой

```
npm init -y
```

- Установите модуль **nodemon**:

```
npm i nodemon --save-dev
```

- Убедитесь, что модуль установился - посмотрите `npm list` или загляните в **07-lab-http-server/package.json**
- Измените файл **package.json** :

```
{
  "name": "07-lab-http-server",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "type": "module",
  "scripts": {
    "start": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

5. Создайте файл **07-lab-http-server/app.js**:

```
import http from 'http';

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html; charset=UTF-8');
  res.end('<h1>Привет, мир!</h1>');
});

server.listen(port, hostname, () => {
  console.log(`Сервер работает на http://${hostname}:${port}/`);
});
```

6. Измените текст в методе `end()` и после изменения сохраните файл в редакторе ``Ctrl + S``:

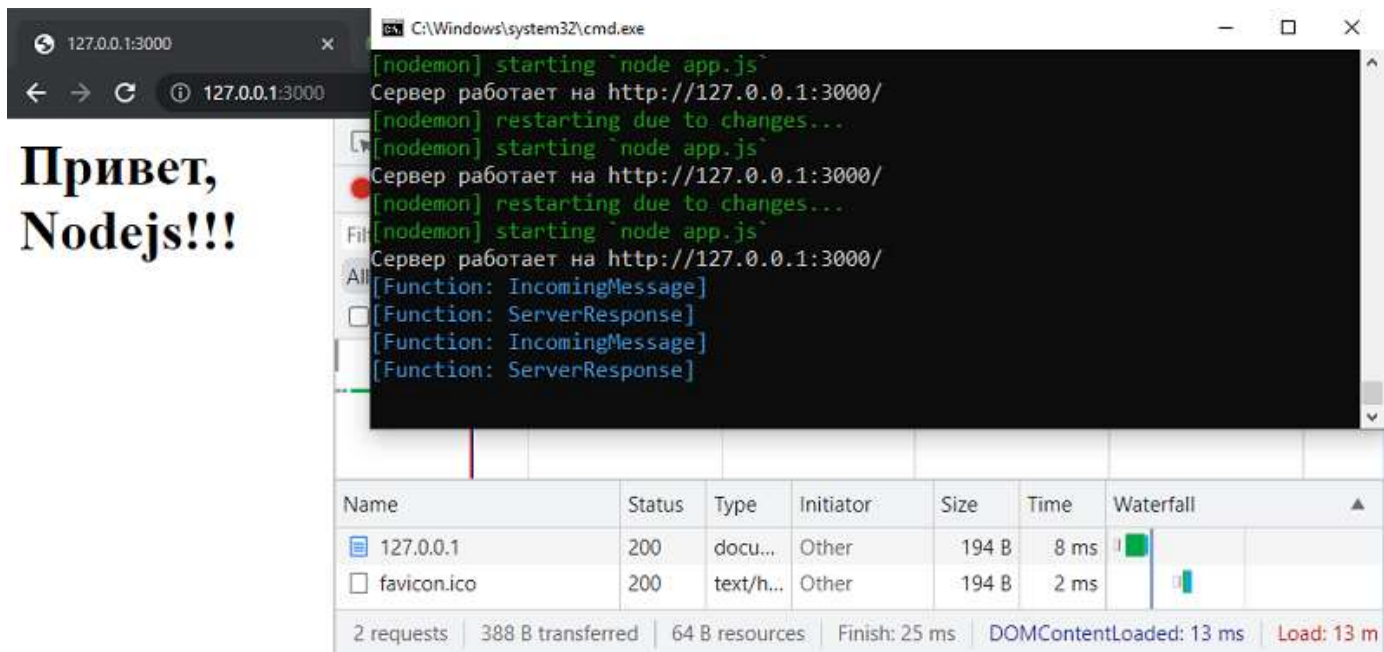
```
res.end('<h1>Привет, Nodejs!!!</h1>');
```

7. Обновите в браузере страницу `http://127.0.0.1:3000/` - вы заметите, что сервер автоматически принял изменения - был перезапущен через **nodemon**.

8. Добавьте в функцию сервера вывод объектов `req` и `res` :

```
console.log(req.constructor)
console.log(res.constructor)
```

9. Убедитесь, что при обновлении страницы в браузере в консоли виден вывод похожий на следующий (По умолчанию браузеры запрашивают файл **favicon.ico**, потому выполняется два запроса, а в консоли видны четыре строки. Если вы хотите, чтобы этого лишнего запроса не было, пропишите в HTML-файле в разделе `head` код `<link rel="icon" href="data:;base64,=">`):



10. Добавьте в функцию сервера вывод свойств объекта `req`, сохраните файл и откройте в браузере страницу <http://127.0.0.1:3000/12>:

```
// console.log(req.headers)
console.log(new URL(req.url, `http://${req.headers.host}`))
console.log(req.httpVersion)
console.log(req.url)
console.log(req.method)
```

```
Сервер работает на http://127.0.0.1:3000/
[Function: IncomingMessage]
[Function: ServerResponse]
URL {
  href: 'http://127.0.0.1:3000/12',
  origin: 'http://127.0.0.1:3000',
  protocol: 'http:',
  username: '',
  password: '',
  host: '127.0.0.1:3000',
  hostname: '127.0.0.1',
  port: '3000',
  pathname: '/12',
  search: '',
  searchParams: URLSearchParams {},
  hash: ''
}
1.1
/12
GET
```

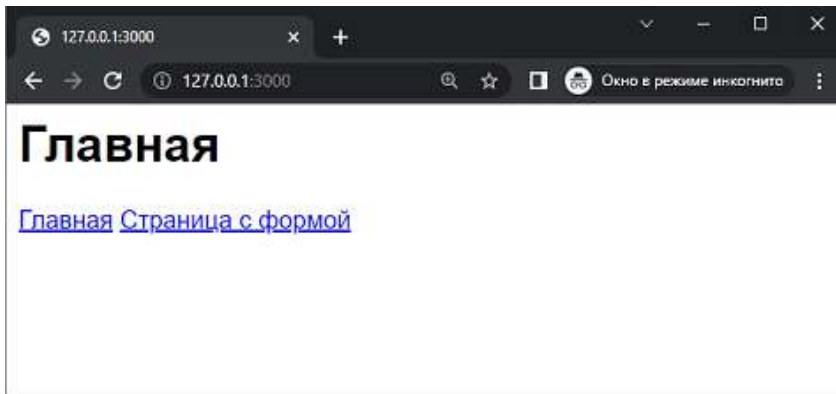
11\*. Познакомьтесь с заметкой <https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/> об анатомии HTTP-транзакции

12. Добавьте в функцию сервера такой код, который отработает при условии GET-запроса на корень сайта `http://127.0.0.1:3000/`:

```

if (req.method === 'GET' && req.url === '/') {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html; charset=UTF-8');
  res.end(`<link rel="icon" href="data:;base64,=">
<style>{*font-family: sans-serif}</style>
<h1>Главная</h1>
<a href="/">Главная</a> <a href="/page">Страница с формой</a><br />
`);
  return;
}

```

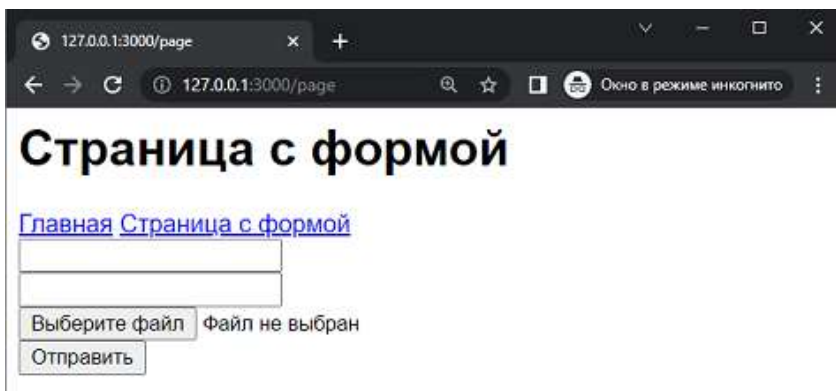


13. Добавьте в функцию сервера такой код, который отработает при условии GET-запроса на адрес `http://127.0.0.1:3000/page` :

```

if (req.method === 'GET' && req.url === '/page') {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html; charset=UTF-8');
  res.end(`<link rel="icon" href="data:;base64,=">
<style>{*font-family: sans-serif}</style>
<h1>Страница с формой</h1>
<a href="/">Главная</a> <a href="/page">Страница с формой</a><br />
<form action="/page" method="post">
  <input type="text" name="firstName" /><br />
  <input type="password" name="pass" /><br />
  <input type="file" name="photo" /><br />
  <input type="hidden" name="param" value='secret' />
  <button>Отправить</button>
</form>
`);
  return;
}

```



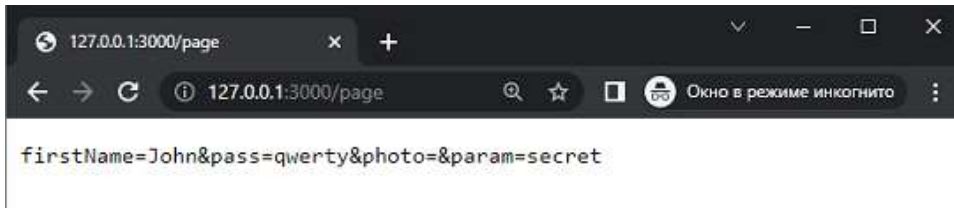
14. Добавьте обработку POST-запроса из формы на адрес `http://127.0.0.1:3000/page` :

```

if (req.method === 'POST' && req.url === '/page') {
  let body = [];
  req.on('data', (chunk) => {
    body.push(chunk);
  }).on('end', () => {
    body = Buffer.concat(body).toString();
    res.end(body);
  });
  return;
}

```

15. Перейдите в браузере на страницу <http://127.0.0.1:3000/page>, заполните форму строками "John" и "qwerty", а затем отправьте форму нажав кнопку **Отправить**. Результат должен быть такой же как на изображении ниже:



- 16 (! обязательно). Попробуйте добавить в форму атрибут `enctype` и изучите результат обработки формы. После эксперимента верните форму в прежнее состояние

```
enctype='multipart/form-data'
```

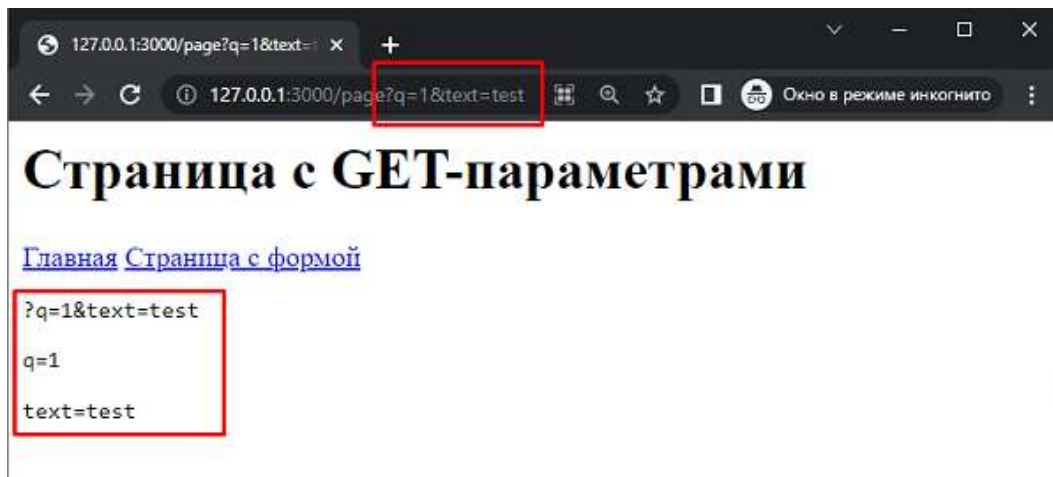
17. Добавьте код, который будет обрабатывать GET-запрос с GET-параметрами

```

// показываем содержимое при GET-запросе с параметрами
// на страницу /page
let url = new URL(req.url, `http://${req.headers.host}`)
if (req.method === 'GET' && url.pathname === '/page') {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html; charset=UTF-8');
  res.end(`<link rel="icon" href="data:;base64,=">
<h1>Страница с GET-параметрами</h1>
<a href="/">Главная</a> <a href="/page">Страница с формой</a><br />
<pre>${url.search}</pre>
<pre>q=${url.searchParams.get('q')}</pre>
<pre>text=${url.searchParams.get('text')}</pre>
`);
  return;
}

```

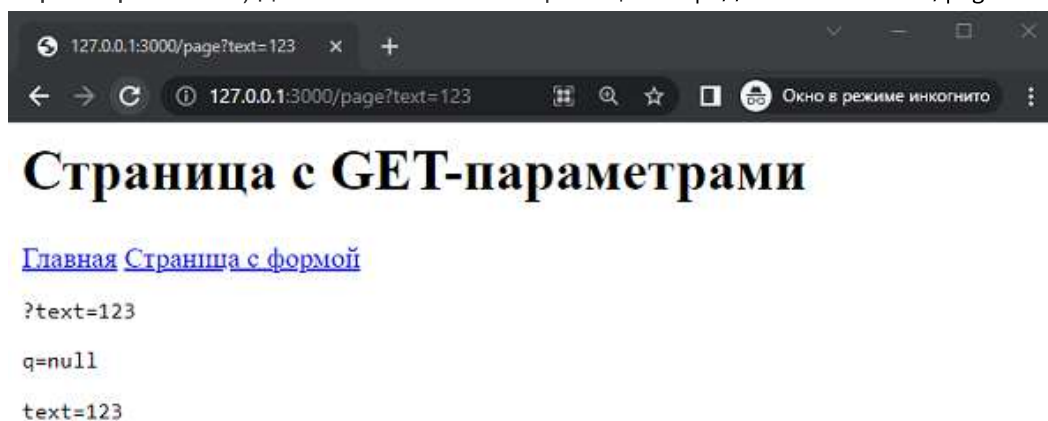
18. В браузере перейдите по адресу <http://127.0.0.1:3000/page?q=1&text=test>. Убедитесь, что GET-параметры **q** и **text** доступны и их значения выводятся на странице. *Примечание: попробуйте задать другие значения GET-параметрам. Например, <http://127.0.0.1:3000/page?q=Lorem&text=Ipsum>*



19. Ниже кода с обработкой POST-запроса, напишите код, который будет получать GET-параметр **id** при запросе по адресу `http://127.0.0.1:3000/addToBasket?id=123` и перенаправлять клиента на страницу `http://127.0.0.1:3000/page?text=123`

```
if (req.method === 'GET' && url.pathname === '/addToBasket') {  
  res.statusCode = 303;  
  const id = parseInt(url.searchParams.get('id'))  
  res.setHeader('Location', 'http://127.0.0.1:3000/page?text='+id);  
  res.end();  
  // полезная работа  
  // ...  
  
  return;  
}
```

20. Перейдите в браузере на страницу `http://127.0.0.1:3000/addToBasket?id=123`, в окне (из-за перенаправления) должна показаться страница `http://127.0.0.1:3000/page?text=123` :



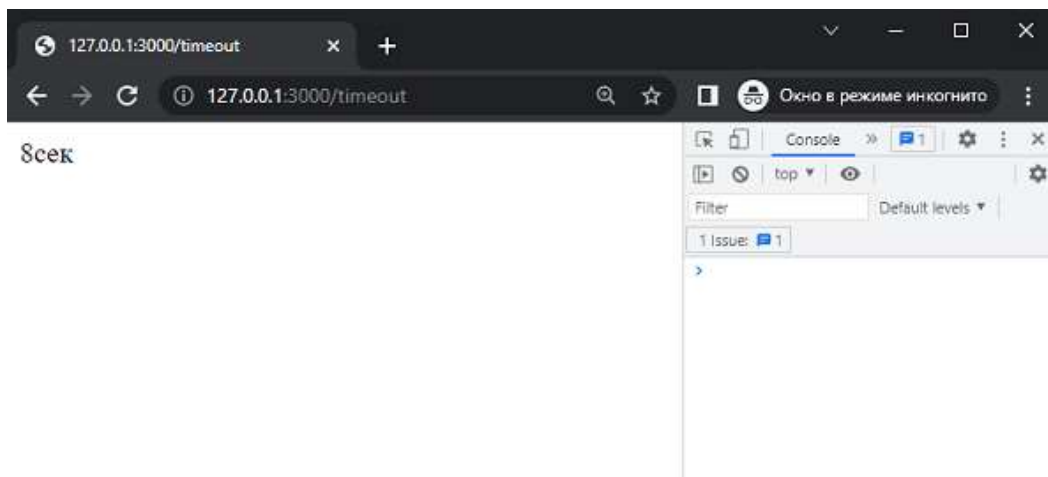
21. Напишите код, который при заходе на страницу по адресу `http://127.0.0.1:3000/timeout` будет через 10 секунд выполнять перенаправление на страницу `http://127.0.0.1:3000/page?text=redirected` (обратите внимание на GET-параметр **text**).  
*Примечание: используйте заголовок 'Refresh:10;url=http://127.0.0.1:3000/page?text=redirected'*

```

if (req.method === 'GET' && url.pathname === '/timeout') {
  res.statusCode = 200;
  res.setHeader('Refresh', '10;url=http://127.0.0.1:3000/page?text=redirected');
  res.setHeader('Content-Type', 'text/html; charset=UTF-8');
  res.end(`
<div id=time>10cek</div>
<script>
let i = 0, steps = 10, duration = 1000;
function tick(){
  if(i < steps - 1){
    i++;
    time.innerHTML = 10 - i + 'cek'
    setTimeout(tick, duration)
  }
}
setTimeout(tick, 0)
</script>
`);

  return;
}

```



22. Напишите код, который при заходе на страницу по адресу `http://127.0.0.1:3000/price` будет заставлять браузер сохранять HTML-файл с названием **price.html** с произвольными данными напоминающими прайс-лист. *Примечание: используйте заголовок 'Content-Disposition: attachment; filename="price.html"'*



```

if (req.method === 'GET' && url.pathname === '/price') {
  res.statusCode = 200;
  res.setHeader('Content-Disposition', 'attachment; filename="price.html"');
  res.setHeader('Content-Type', 'text/html; charset=UTF-8');

  let items = [
    {id: 123, title: 'Товар1', price: 1000},
    {id: 124, title: 'Товар2', price: 2000},
    {id: 127, title: 'Товар3', price: 3000},
  ]
  let content = ''

  items.forEach( item => content += `<div>
    <h3>${item.title}</h3>
    <p>Цена: ${item.price}</p>
  </div>`)

  res.end(content);

  return;
}

```



# Главная

[Главная](#) [Страница с формой](#)



23. Остановите сервер `ctrl + c`, установить **formidable** и запустите сервер снова

```

npm install formidable@v3
npm start

```

24. Пропишите импорт в верхней части файла **07-lab-http-server/app.js**

```

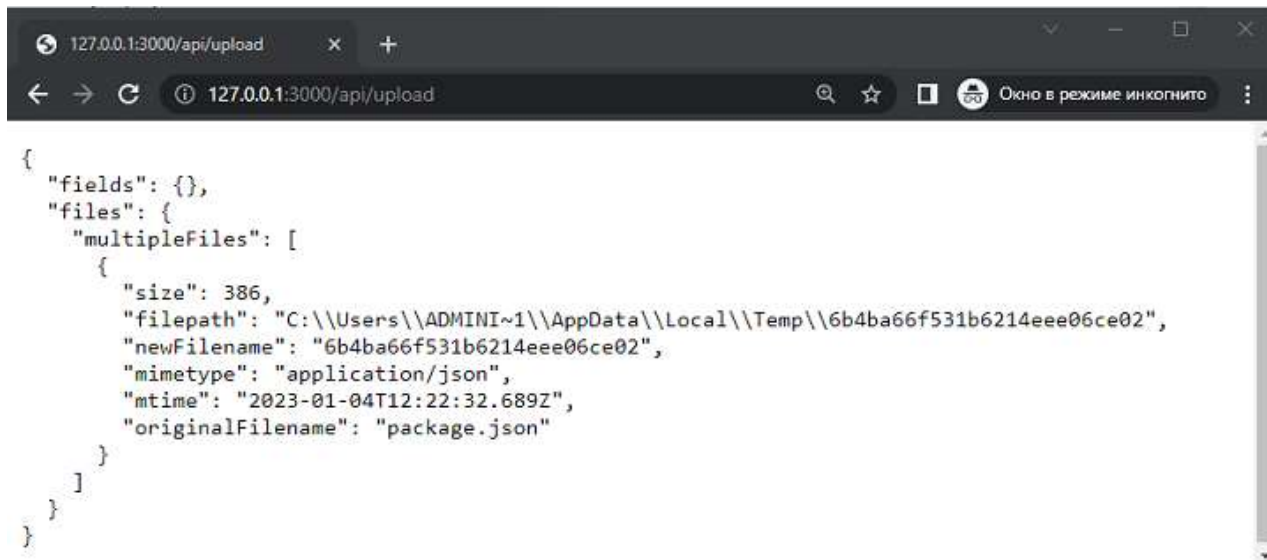
import formidable from 'formidable';

```

25. Познакомьтесь с примером работы модуля **formidable** на странице

<https://www.npmjs.com/package/formidable#user-content-examples> и адаптируйте его для текущего приложения: при GET запросе на адрес `http://127.0.0.1/api/upload` должна показываться форма загрузки файла, а при POST-запросе - отображаться данные формы и/или информация о загружаемых данных





The screenshot shows a web browser window with the address bar displaying `127.0.0.1:3000/api/upload`. The browser is in incognito mode, as indicated by the text "Окно в режиме инкогнито" in the address bar. The main content area displays a JSON object representing an uploaded file:

```
{
  "fields": {},
  "files": {
    "multipleFiles": [
      {
        "size": 386,
        "filepath": "C:\\Users\\ADMINI~1\\AppData\\Local\\Temp\\6b4ba66f531b6214eee06ce02",
        "newFilename": "6b4ba66f531b6214eee06ce02",
        "mimetype": "application/json",
        "mtime": "2023-01-04T12:22:32.689Z",
        "originalFilename": "package.json"
      }
    ]
  }
}
```

26. Реализуйте обработку GET-запроса на адрес `http://127.0.0.1/users` - при открытии страницы, нужно показать список пользователей с фейкового URL `https://jsonplaceholder.typicode.com/users`. *Примечание: получить данные с фейкового сервиса нужно при помощи [Fetch API](#), но на стороне Nodejs. Не забудьте пометить функцию сервера асинхронной, тогда при обработке запроса можно будет использовать ключевое слово **await***