



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по лабораторной работе № 3

Дисциплина: Архитектура ЭВМ

Студент

ИУ7-52Б

(Группа)

(Подпись, дата)

В.А. Иванов

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

А.Ю. Попов

(И.О. Фамилия)

Москва, 2020

Цель работы

Целью лабораторной работы освоение методов работы с запросами и шаблонизаторами.

Задание 5.1

Создать сервер. Сервер должен выдавать страницу с тремя текстовыми полями и кнопкой. В поля ввода вбивается информация о почте, фамилии и номере телефона человека. При нажатии на кнопку "*Отправить*" введённая информация должна отправляться с помощью **POST** запроса на сервер и добавляться к концу файла (в файле накапливается информация). При этом **на стороне сервера** должна происходить проверка: являются ли почта и телефон уникальными. Если они уникальны, то идёт добавление информации в файл. В противном случае добавление не происходит. При отправке ответа с сервера клиенту должно приходить сообщение с информацией о результате добавления (добавилось или не добавилось). Результат операции должен отображаться на странице.

Программная реализация

index.js (сервер)

```
"use strict";

const express = require("express");
const fs = require("fs");
const file_name = "users.txt"

const app = express();
const port = 5000;
app.listen(port);
console.log(`Server on port ${port}`);

const way = __dirname + "/static";
app.use(express.static(way));

// заголовки в ответ клиенту
app.use(function(req, res, next) {
  res.header("Cache-Control", "no-cache, no-store, must-revalidate");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  res.header("Access-Control-Allow-Origin", "*");
  next();
});

// body
function loadBody(request, callback) {
  let body = [];
  console.log(body);
  request.on('data', (chunk) => {
    body.push(chunk);
  }).on('end', () => {
    body = Buffer.concat(body).toString();
    callback(body);
  });
}

function get_f_arr(f_name)
{
  let f_arr;
  if (fs.existsSync(f_name)) {
```

```

        const f_str = fs.readFileSync(f_name, "utf-8");
        if (f_str === "")
            f_arr = [];
        else
            f_arr = JSON.parse(f_str);
    } else {
        f_arr = [];
    }
    return f_arr
}

app.post("/save/info", function(request, response) {
    loadBody(request, function(body) {
        const obj = JSON.parse(body);
        let f_arr = get_f_arr(file_name);

        let cont_flag = false;
        for (let i=0; i<f_arr.length && !cont_flag; i++) {
            if (f_arr[i].email == obj.email)
                cont_flag = true;
            else if (f_arr[i].phone == obj.phone)
                cont_flag = true;
        }

        let msg;
        if (cont_flag) {
            msg = "Entry didn't added";
        } else {
            msg = "Entry added";
            f_arr.push(obj);
            fs.writeFileSync(file_name, JSON.stringify(f_arr));
        }

        response.end(JSON.stringify({ result: msg }));
    });
});

```

req_code.js (клиент)

```

"use strict";

function ajaxPost(urlString, bodyString, callback) {
    let r = new XMLHttpRequest();
    r.open("POST", urlString, true);
    r.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
    r.send(bodyString);
    r.onload = function() {
        callback(r.response);
    }
}

window.onload = function() {
    // input fields
    const f_email = document.getElementById("field-email");
    const f_surname = document.getElementById("field-surname");
    const f_phone = document.getElementById("field-phone");

    // button
    const btn = document.getElementById("send-btn");

    // click event
    btn.onclick = function() {
        const msg = JSON.stringify({
            email: f_email.value,
            surname: f_surname.value,

```

```

        phone: f_phone.value
    });

    ajaxPost("/save/info", msg, function(answerString) {
        const answerObject = JSON.parse(answerString);
        const result = answerObject.result;
        alert(result);
    });
});
};
};

```

Тесты

Ввод:

Task 5

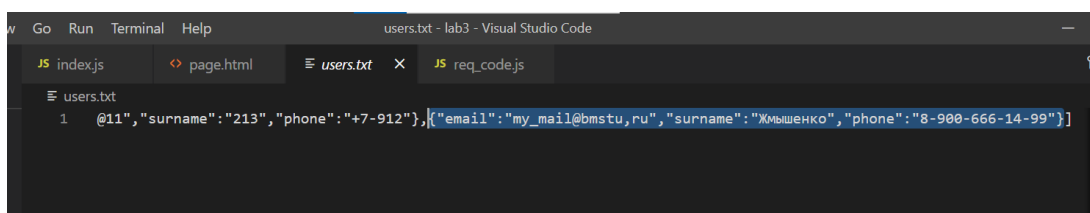
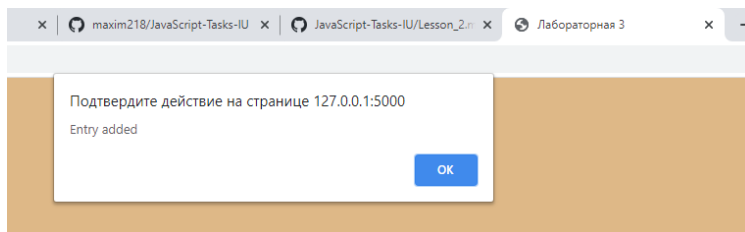
Почта

Фамилия

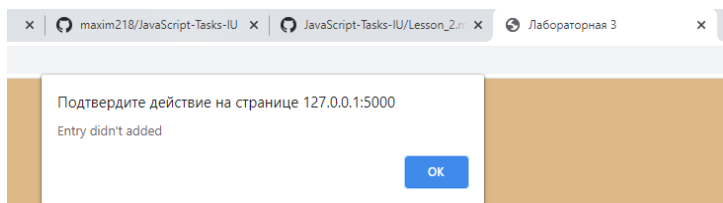
Номер телефона

Отправить

Вывод:



При повторном запросе:



Задание 5.2

Добавить серверу возможность отправлять клиенту ещё одну страницу. На данной странице должно быть поле ввода и кнопка. В поле ввода вводится почта человека. При нажатии на кнопку "*Отправить*" на сервер отправляется **GET** запрос. Сервер в ответ на **GET** запрос должен отправить информацию о человеке с данной почтой в формате **JSON** или сообщение об отсутствии человека с данной почтой.

Программная реализация

Дополнение index.js (сервер)

```
function search_email(email) {
    let obj = null;
    const f_arr = get_f_arr(file_name);
    for (let i=0; i<f_arr.length && !obj; i++) {
        if (f_arr[i].email == email)
            obj = f_arr[i];
    }
    return obj;
}

app.get("/search", function(request, response) {
    const email = request.query.email;
    const res_obj = search_email(email);
    let answer = {is_found : true, obj:res_obj};
    if (res_obj === null)
        answer.is_found = false;

    response.end(JSON.stringify(answer));
});
```

search.js (клиент)

```
"use strict";

function ajaxGet(urlString, callback) {
    let r = new XMLHttpRequest();
    r.open("GET", urlString, true);
    r.setRequestHeader("Content-Type", "text/plain;charset=UTF-8");
    r.send(null);
    r.onload = function() {
        callback(r.response);
    };
};

window.onload = function() {
    // input fields
    const f_email = document.getElementById("field-email");
    // button
    const btn = document.getElementById("send-btn");
    // Output fields
    const out_email = document.getElementById("result-email");
    const out_surname = document.getElementById("result-surname");
    const out_phone = document.getElementById("result-phone");

    function show_user(obj) {
        out_email.innerHTML = `Почта:      ${obj.email}`;
        out_surname.innerHTML = `Фамилия:   ${obj.surname}`;
        out_phone.innerHTML = `Телефон:   ${obj.phone}`;
    }
}
```

```

    }

    function show_void() {
        out_email.innerHTML = ``;
        out_surname.innerHTML = ``;
        out_phone.innerHTML = ``;
    }

    // click event
    btn.onclick = function() {
        const email = f_email.value;
        const url = `/search?email=${email}`;

        ajaxGet(url, function(stringAnswer) {
            const objectAnswer = JSON.parse(stringAnswer);
            if (objectAnswer.is_found)
                show_user(objectAnswer.obj);
            else {
                show_void();
                alert(`Пользователь с почтой ${email} не найден`);
            }
        });
    };
};
obj

```

Тесты

Поиск человека по почте

Введите почту:

Найти

Результат поиска:

Почта: @11
 Фамилия: 213
 Телефон: +7-912

← → ↻ 🏠 ⓘ 127.0.0.1:5000/search.html

Поиск человека по почте

Введите почту:

Найти

Подтвердите действие на странице 127.0.0.1:5000
 Пользователь с почтой @12 не найден

ОК

Задание 3.3

С клавиатуры считывается строка - название расширения файлов. Далее считывается строка - адрес папки. Необходимо перебрать все файлы в папке и вывести содержимое файлов, у которых расширение совпадает с введенным расширением.

Программная реализация

```
"use strict";

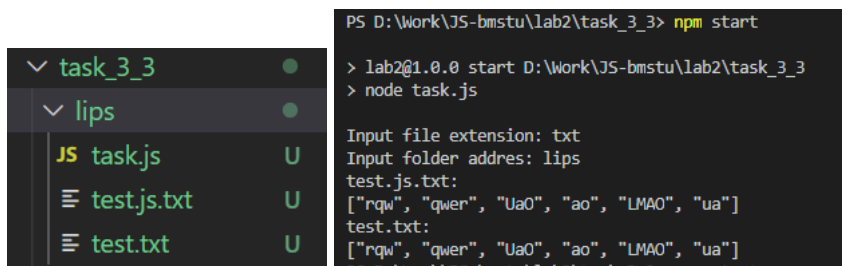
const readlineSync = require('readline-sync');
const fs = require("fs");
const ext = readlineSync.question("Input file extension: ");
const addr = readlineSync.question("Input folder address: ");

if (fs.existsSync(addr)) {
    const file_arr = fs.readdirSync(addr);

    file_arr.forEach(f_name => {
        let parts = f_name.split('.');
        if (parts[parts.length-1] == ext) {
            console.log(f_name + ":");

            let str = fs.readFileSync(addr+"\\ "+f_name, "utf-8");
            if (str == "") {
                console.log("Empty file");
            } else {
                console.log(str);
            }
        }
    });
} else {
    console.log("Folder not exists");
}
```

Тесты



The image shows a file explorer on the left and a terminal window on the right. The file explorer displays a directory structure with 'task_3_3' expanded, showing 'lips' and 'task.js'. The terminal window shows the command 'npm start' and the output of the program, which prompts for 'Input file extension: txt' and 'Input folder address: lips', then lists the contents of 'test.js.txt' and 'test.txt'.

```
PS D:\Work\JS-bmstu\lab2\task_3_3> npm start
> lab2@1.0.0 start D:\Work\JS-bmstu\lab2\task_3_3
> node task.js

Input file extension: txt
Input folder address: lips
test.js.txt:
["rqw", "qwer", "Ua0", "ao", "LMA0", "ua"]
test.txt:
["rqw", "qwer", "Ua0", "ao", "LMA0", "ua"]
```


Задание 3.4

Дана вложенная структура файлов и папок. Все файлы имеют расширение "txt". Необходимо рекурсивно перебрать вложенную структуру и вывести имена файлов, у которых содержимое не превышает по длине 10 символов.

Программная реализация

```
"use strict";

const fs = require("fs");

function lookup_folder(addr){
  if (!fs.existsSync(addr)) { return; }

  const file_arr = fs.readdirSync(addr);
  file_arr.forEach(f_name => {
    let parts = f_name.split('.');

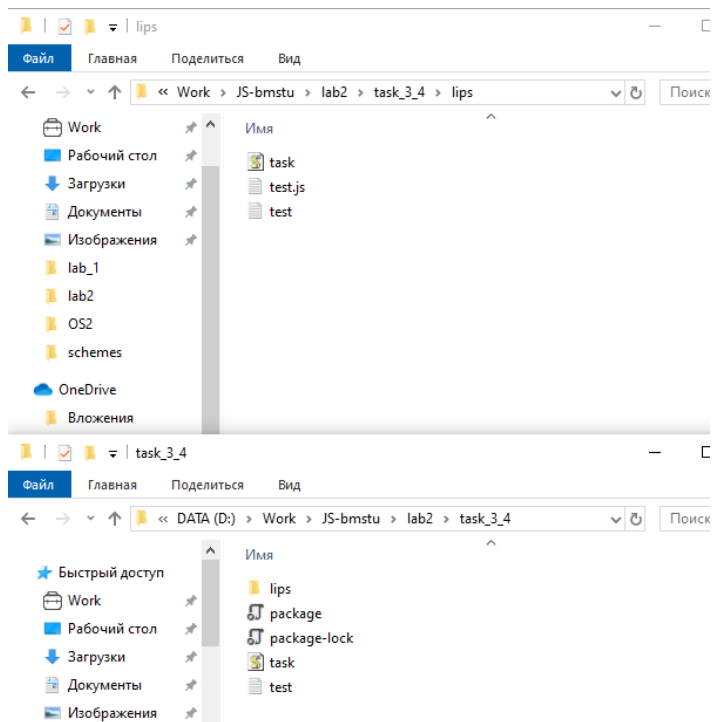
    if (parts.length == 1) {
      lookup_folder(addr+"\\"+f_name);
    } else {
      let str = fs.readFileSync(addr+"\\"+f_name, "utf-8");

      if (str.length <= 10) {
        console.log(addr+"\\"+f_name);
        console.log(str);
      }
    }
  });
}

lookup_folder(".");
```

Тесты

Ввод:



Вывод:

```
PS D:\Work\JS-bmstu\lab2\task_3_4> npm start

> lab2@1.0.0 start D:\Work\JS-bmstu\lab2\task_3_4
> node task.js

.\lips\task.js
"use";
.\lips\test.txt
["rqw"]
```

Задание 3.5

С клавиатуры считывается число N. Далее считывается N строк - имена текстовых файлов. Необходимо склеить всё содержимое введенных файлов в одну большую строку и сохранить в новый файл.

Программная реализация

```
"use strict";

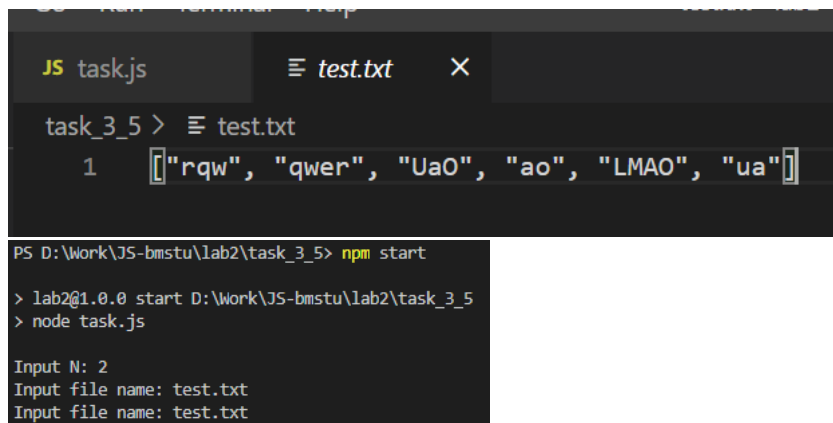
const fs = require("fs");
const readlineSync = require('readline-sync');

let result_str = "";
const n = parseInt(readlineSync.question("Input N: "));

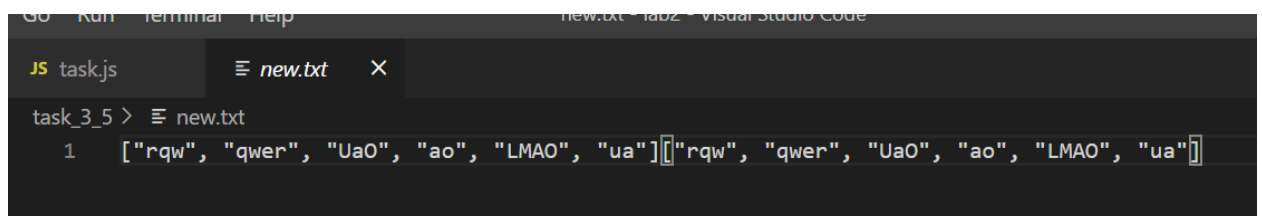
for (let i=0; i<n; i++){
    let f_name = readlineSync.question("Input file name: ");
    if (fs.existsSync(f_name)) {
        let str = fs.readFileSync(f_name, "utf-8");
        result_str += str;
    } else {
        console.log("File not exists");
    }
}
fs.writeFileSync("new.txt", result_str);
```

Тесты

Ввод:



Вывод:



Задание 3.6

Написать код, который позволяет определить максимальный возможный уровень вложенности друг в друга полей в объекте, чтобы данный объект можно было преобразовать в строку формата JSON. Ответом является целое число.

Программная реализация

```
"use strict";

const e = require("express");

class Box {
  constructor (depth) {
    this.d = depth;
    if (depth > 0)
      this.next = new Box(depth-1);
    else
      this.next = null;
  }
}

let size = 100;
let step = 128;
while (step > 1) {
  try {
    let b = new Box(size);
    let jsonStr = JSON.stringify(b);
    size += step;
  } catch (RangeError) {
    size -= step;
    step /= 2;
  }
}
console.log(size);
```

Тесты

```
PS D:\Work\JS-bmstu\lab2\task_3_6> npm start

> lab2@1.0.0 start D:\Work\JS-bmstu\lab2\task_3_6
> node task.js

964
PS D:\Work\JS-bmstu\lab2\task_3_6> |
```

Задание 3.7

Из файла считывается строка в формате JSON. В этой строке информация об объекте, в котором находится большое количество вложенных друг в друга полей. Объект представляет из себя дерево. Необходимо рекурсивно обработать дерево и найти максимальную вложенность в дереве. Необходимо вывести на экран ветку с максимальной вложенностью.

Программная реализация

```
"use strict";

const fs = require("fs");
const f_name = "json-data.txt";

function create_file() {
    const probability = 0.53;
    class Box {
        constructor (depth) {
            this.d = depth;
            this.leaf1 = null;
            this.leaf2 = null;
            if (Math.random() > 1 - probability)
                this.leaf1 = new Box(depth+1);
            if (Math.random() > 1 - probability)
                this.leaf2 = new Box(depth+1);
        }
    }
    let b = new Box(0);
    let jStr = JSON.stringify(b);
    console.log(jStr);
    fs.writeFileSync(f_name, jStr);
}

function path_lenght(path) {
    return (path.match(new RegExp("/", "g")) || []).length
}

function process_tree(tree) {
    let path = "";
    for (let key in tree) {
        if (typeof(tree[key]) == "object" && tree[key] != null) {
            let t_path = key + "/" + process_tree(tree[key]);
            if (path_lenght(t_path) > path_lenght(path))
                path = t_path;
        }
    }
    return path;
}
```

```
// create_file();

let jStr = fs.readFileSync(f_name, "utf-8");
console.log("Scaned JSON-data: " + jStr);

let original_tree = JSON.parse(jStr);
let path = process_tree(original_tree);
let path_l = path_lenght(path);

console.log("Path: " + path.slice(0, path.length-1));
console.log("Length: " + path_l);
```

Тесты

Ввод:

Для проверки работоспособности программы была создана функция, записывающая в файл древовидную структуру в формате JSON. В тесте использована строка:

```
{ "d":0,"leaf1":null,"leaf2":{"d":1,"leaf1":{"d":2,"leaf1":null,"leaf2":{"d":3,"leaf1":null,"leaf2":{"d":4,"leaf1":{"d":5,"leaf1":{"d":6,"leaf1":{"d":7,"leaf1":null,"leaf2":null},"leaf2":null},"leaf2":{"d":6,"leaf1":{"d":7,"leaf1":{"d":8,"leaf1":{"d":9,"leaf1":null,"leaf2":{"d":10,"leaf1":{"d":11,"leaf1":null,"leaf2":null},"leaf2":null},"leaf2":{"d":9,"leaf1":null,"leaf2":null},"leaf2":null},"leaf2":{"d":5,"leaf1":null,"leaf2":null}}}},"leaf2":null}},"leaf2":null}},"leaf2":{"d":5,"leaf1":null,"leaf2":null}}}},"leaf2":null}}
```

Вывод:

```
Scaned JSON-data: {"d":0,"leaf1":null,"leaf2":{"d":1,"leaf1":{"d":2,"leaf1":null,"leaf2":{"d":3,"leaf1":null,"leaf2":{"d":4,"leaf1":{"d":5,"leaf1":{"d":6,"leaf1":{"d":7,"leaf1":null,"leaf2":null},"leaf2":null},"leaf2":{"d":6,"leaf1":{"d":7,"leaf1":{"d":8,"leaf1":{"d":9,"leaf1":null,"leaf2":{"d":10,"leaf1":{"d":11,"leaf1":null,"leaf2":null},"leaf2":null},"leaf2":{"d":9,"leaf1":null,"leaf2":null},"leaf2":null},"leaf2":{"d":5,"leaf1":null,"leaf2":null}}}},"leaf2":null}},"leaf2":{"d":5,"leaf1":null,"leaf2":null}}}},"leaf2":null}},"leaf2":{"d":5,"leaf1":null,"leaf2":null}}}},"leaf2":null}}

Path: leaf2/leaf1/leaf2/leaf2/leaf1/leaf2/leaf1/leaf1/leaf1/leaf2/leaf1
Length: 11
```

Задание 4.1

Запустить сервер. Реализовать на сервере функцию для сравнения трёх чисел и выдачи наибольшего из них. Реализовать страницу с формой ввода для отправки запроса на сервер.

Программная реализация

```
"use strict";
const fs = require("fs");
const express = require("express");

const app = express();
const port = 5015;
const query_page = "a.html"
app.listen(port);

app.get("/me/page", function(request, response) {
  const nameString = request.query.p;
  let contentString;
  if (fs.existsSync(query_page)) {
    contentString = fs.readFileSync(query_page, "utf8");
  } else {
    contentString = "Page is not available";
  }
  response.end(contentString);
});

app.get("/calculate/sum", function(request, response) {
  const a = request.query.a;
  const b = request.query.b;
  const c = request.query.c;

  const aInt = parseInt(a);
  const bInt = parseInt(b);
  const cInt = parseInt(c);

  let answer;
  if (aInt !== NaN && bInt !== NaN && cInt !== NaN) {
    const sInt = Math.max(aInt, bInt, cInt);
    answer = "Max number: " + sInt;
  } else {
    answer = "Incorrect input";
  }

  response.end(answer);
});
```

Тесты

Ввод:

Введите 3 числа

1:

2:

3:

Вывод:



localhost:5015/calculate/sum?a=3&b=2&c=5

Max number: 5

Задание 4.2

Запустить сервер. На стороне сервера должен храниться файл, внутри которого находится JSON строка. В этой JSON строке хранится информация о массиве объектов. Реализовать на сервере функцию, которая принимает индекс и выдает содержимое ячейки массива по данному индексу.

Реализовать страницу с формой ввода для отправки запроса на сервер.

Программная реализация

```
"use strict";
const fs = require("fs");
const express = require("express");

const app = express();
const port = 5015;
const query_page = "a.html"
app.listen(port);

app.get("/me/page", function(request, response) {
  const nameString = request.query.p;
  let contentString;
  if (fs.existsSync(query_page)) {
    contentString = fs.readFileSync(query_page, "utf8");
  } else {
    contentString = "Page is not available";
  }
  response.end(contentString);
});

app.get("/show_i", function(request, response) {
  const jStr = JSON.stringify([1, 2, 3, 4, 5, 6, 7, 8]);
  const i = parseInt(request.query.i);

  const obj = JSON.parse(jStr);
  let answer;

  if (typeof(obj) == 'object' && obj != null) {
    if (i < obj.length) {
      answer = "" + obj[i];
    } else {
      answer = "Index is out of array range";
    }
  } else {
    console.log(typeof(obj));
    answer = "JSON string doesn't contain an array";
  }

  response.end(answer);
});
```

Тесты

Ввод:

Введите индекс	Введите индекс
i:	i:
<input type="text" value="1"/>	<input type="text" value="100"/>
<input type="button" value="Отправить запрос"/>	<input type="button" value="Отправить запрос"/>

Вывод:

--	--

Задание 4.3

Написать программу, которая на вход получает массив названий полей и адрес запроса (куда отправлять). Программа должна генерировать HTML разметку страницы, в которую встроена форма для отправки запроса.

Программная реализация

```
"use strict";

const readlineSync = require('readline-sync');
const fs = require('fs');
const express = require('express');
const html_name = 'gen.html';

function html_str(addr, field_arr) {
    let s = "";

    s +=
    '<!DOCTYPE html>\n\
    <html>\n\
    <head>\n\
        <meta charset="UTF-8">\n\
        <title>Страница A</title>\n\
    </head>\n';

    s +=
    '<body>\n\
        <h1>Сгенерированное поле запроса</h1>\n\
        <form method="GET" action="' + addr + '">\n';

    for (let i=0; i < field_arr.length; i++) {
        s += '\t<p>' + field_arr[i] + '</p>\n\
        <input name="' + field_arr[i] + '" spellcheck="false" autocomplete="off">\n';
    }

    s += '\t<input type="submit" value="Отправить запрос">\n\
    </form>\n\
    </body>\n\
    </html>';

    return s;
}

function generate_html(addr, field_arr) {
    const hStr = html_str(addr, field_arr);
    console.log(hStr);

    fs.writeFileSync(html_name, hStr);
}
```

```

function show_html() {
  const app = express();
  const port = 5015;
  app.listen(port);

  app.get('/me/page', function(request, response) {
    let contentString;
    if (fs.existsSync(html_name)) {
      contentString = fs.readFileSync(html_name, "utf8");
    } else {
      contentString = "Page is not available";
    }
    response.end(contentString);
  });
}

const req_addr = readlineSync.question("Input address: ")
const field_arr = readlineSync.question("Input fields (separated with ,): ").split(", ");

generate_html(req_addr, field_arr);
show_html();

```

Тесты

Ввод:

```

PS D:\Work\JS-bmstu\lab2\task_4_3> npm start

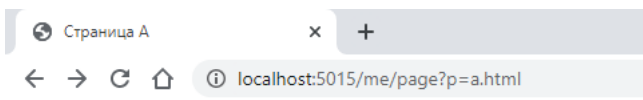
> lab2@1.0.0 start D:\Work\JS-bmstu\lab2\task_4_3
> node task.js

Input address: love/js
Input fields (separated with ,): a, q, with space

```

Вывод:

```
JS task.js ● gen.html X
task_4_3 > gen.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Страница A</title>
6  </head>
7  <body>
8      <h1>Сгенерированное поле запроса</h1>
9      <form method="GET" action="love/js">
10         <p>a</p>
11         <input name="a" spellcheck="false" autocomplete="off">
12         <p>q</p>
13         <input name="q" spellcheck="false" autocomplete="off">
14         <p>with space</p>
15         <input name="with space" spellcheck="false" autocomplete="off">
16         <input type="submit" value="Отправить запрос">
17     </form>
18 </body>
19 </html>
```



Сгенерированное поле запроса

a

q

with space

Задание 4.4

Запустить сервер. Реализовать на сервере функцию, которая принимает на вход числа А, В и С. Функция должна выдавать массив целых чисел на отрезке от А до В, которые делятся на С нацело.

Программная реализация

```
"use strict";
const fs = require("fs");
const express = require("express");

const app = express();
const port = 5015;
const query_page = "a.html"
app.listen(port);
console.log("Server is running");

app.get("/me/page", function(request, response) {
  const nameString = request.query.p;
  let contentString;
  if (fs.existsSync(query_page)) {
    contentString = fs.readFileSync(query_page, "utf8");
  } else {
    contentString = "Page is not available";
  }
  response.end(contentString);
});

app.get("/show/interval", function(request, response) {
  const a = parseInt(request.query.a);
  const b = parseInt(request.query.b);
  const c = parseInt(request.query.c);
  let answer;

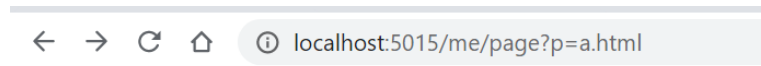
  console.log(a, b, c);
  if (a == NaN || b == NaN || c == NaN) {
    answer = "Incorrect input";
  } else if (a > b) {
    answer = "Incorrect range";
  } else if (c < 1) {
    answer = "Incorrect c value";
  } else {
    let arr = []
    console.log(a / c);
    let i = parseInt(a / c) * c;
    if (i < a) i += c;
    console.log(i);
    for (; i <= b; i += c)
      arr.push(i);
  }
});
```

```
    console.log(arr);

    answer = "Array: " + JSON.stringify(arr);
  }
  response.end(answer);
});
```

Тесты

Ввод:



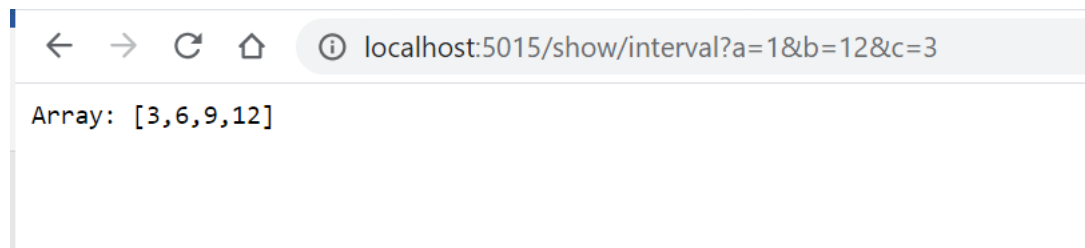
Введите числа А, В и С

А:

В:

С:

Вывод:



Вывод

В рамках лабораторной работы было выполнено ознакомление и практическое закрепление основ работы с JSON, потоками ввода/вывода, управления серверами в языке JS.