|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 3 |

**Дисциплина:** Архитектура ЭВМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | В.А. Иванов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | А.Ю. Попов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**Цель работы**

Целью лабораторной работы освоение методов работы с AJAX запросами, шаблонизаторами и cookie.

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);

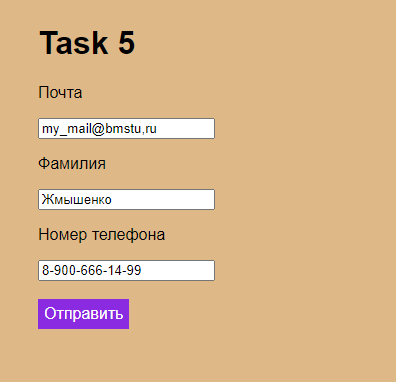
        });

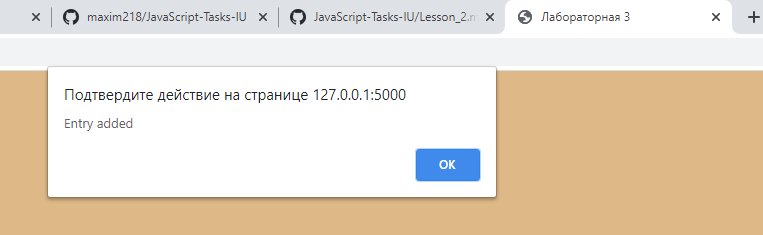
    };

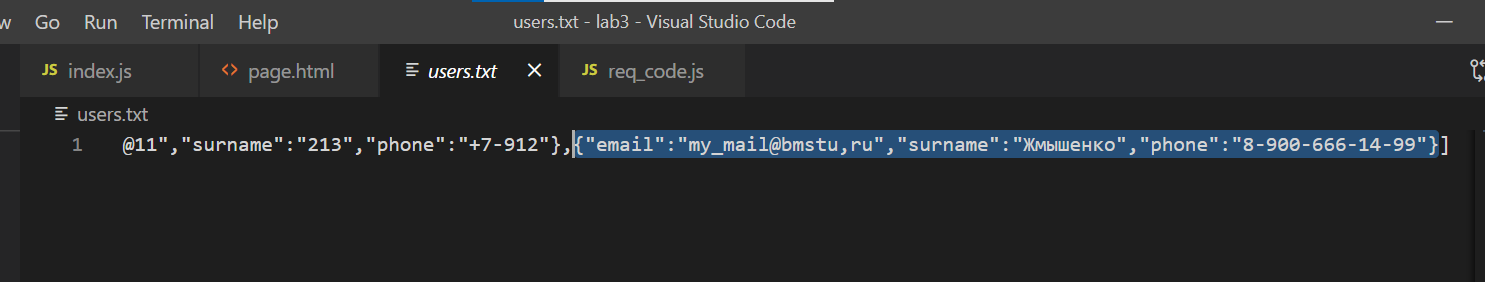
};

**Тесты**

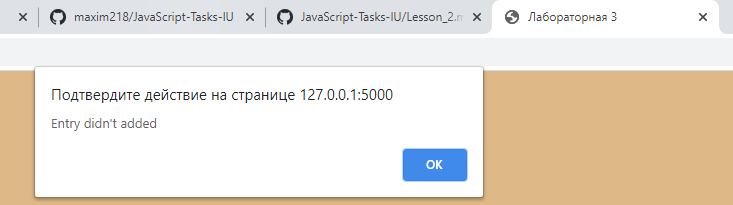
Ввод:

  
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`);

            }

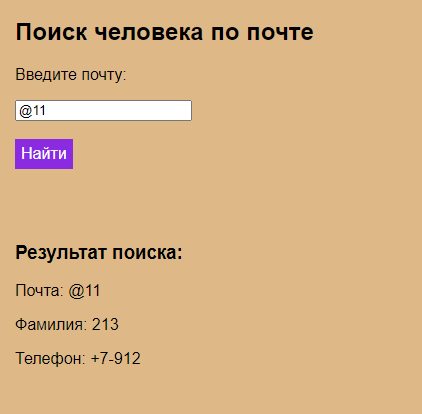
        });

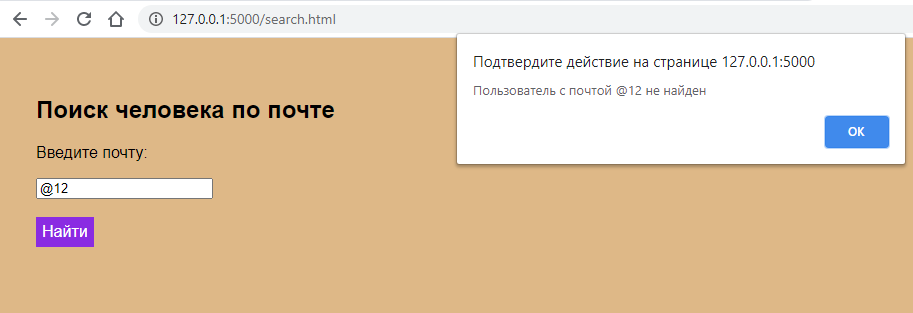
    };

};

            obj

**Тесты**





Задание 5.3

Оформить внешний вид созданных страниц с помощью **CSS**. Информация со стилями **CSS** для каждой страницы должна храниться в отдельном файле. Стили **CSS** должны быть подключены к страницам.

**Программная реализация**

**search.html**

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Лабораторная 3</title>

    <link rel="stylesheet" href="/search.css">

</head>

<body>

    <h2>Поиск человека по почте</h2>

    <p>Введите почту:</p>

    <input id="field-email" type="text" spellcheck="false" autocomplete="off">

    <br>

    <br>

    <div id="send-btn" class="btn-class">Найти</div>

    <br>

    <br>

    <h3>Результат поиска:</h3>

    <p id="result-email"></p>

    <p id="result-surname"></p>

    <p id="result-phone"></p>

    <script src="/search.js"></script>

</body>

</html>

**search.css**

body {

    margin-inline-end: auto;

    background: rgb(237, 238, 240);

    font-family: Geneva, Arial, Helvetica, sans-serif;

}

.btn-class {

    padding: 6px;

    background: rgb(74, 118, 168);

    color: white;

    cursor: help;

    display: inline-block;

}

**page.html**

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Лабораторная 3</title>

    <link rel="stylesheet" href="/page.css">

</head>

<body>

    <h1>Задание 5</h1>

    <p>Почта</p>

    <input id="field-email" type="text" spellcheck="false" autocomplete="off">

    <br>

    <p>Фамилия</p>

    <input id="field-surname" type="text" spellcheck="false" autocomplete="off">

    <br>

    <p>Номер телефона</p>

    <input id="field-phone" type="text" spellcheck="false" autocomplete="off">

    <br>

    <br>

    <div id="send-btn" class="btn-class">Отправить</div>

    <script src="/req\_code.js"></script>

</body>

</html>

**page.css**

body {

    padding: 15px;

    background: rgb(235, 239, 243);

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

    float: left;

    margin: 3px;

}

.btn-class {

    padding: 15px;

    background: rgb(63, 68, 72);

    color: rgb(200, 200, 204);

    cursor: pointer;

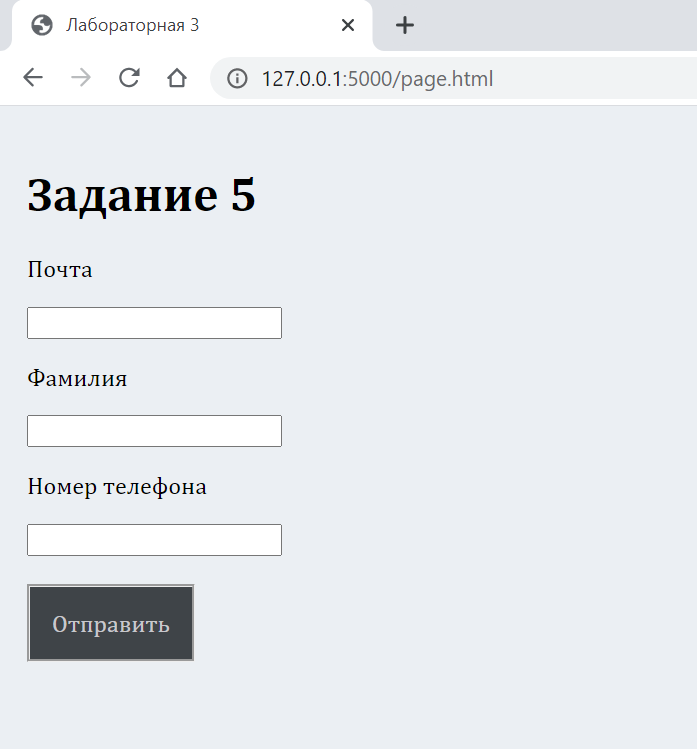
    display: inline-table;

    border-style: groove;

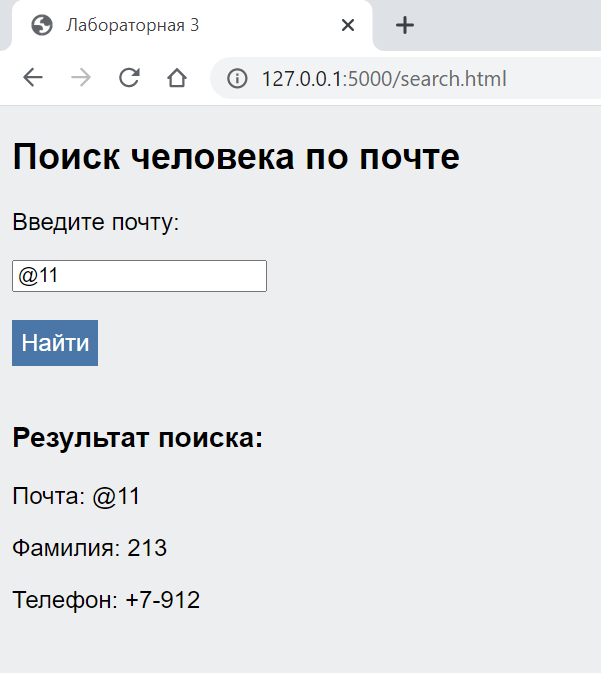
}

**Тесты**

**page.html**

****

**search.html**

****

Задание 6.1

Создать сервер. В оперативной памяти на стороне сервера создать массив, в котором хранится информация о компьютерных играх (название игры, описание игры, возрастные ограничения). Создать страницу с помощью шаблонизатора. В **url** передаётся параметр возраст (целое число). Необходимо отображать на этой странице только те игры, у которых возрастное ограничение меньше, чем переданное в **url** значение.

**Программная реализация**

**index.js (сервер)**

"use strict";

const gameArr = [];

function addGame(name\_, desr\_, limit\_) {

    if (limit\_ < 0)

        return;

    let game = {

        name: name\_,

        description: desr\_,

        age\_limit: limit\_

    };

    gameArr.push(game);

}

function fillGameArr(){

    addGame("LocoRoco", "Платформер, аркада и головоломка с оригинальными графическими решениями", 0);

    addGame("Crash Bandicoot", "Woah!", 6);

    addGame("Baldur's Gate 3", "Заставляют убивать людей, полуросликов и танцевать нагишом, кошмар", 17);

    addGame("Gulman 5", "Gulman - герой, Superman - нет", 16);

    addGame("Pretty Neko", "Оно того не стоит, не лезь", 21);

    addGame("Pretty Angel", "см. описание Pretty Neko", 21);

    addGame("Dota 2", "2 sides, 3 lines, 100 hours", 16);

}

function getGames(age) {

    let ansArr = [];

    for (let i=0; i < gameArr.length; i++)

        if (gameArr[i].age\_limit <= age)

            ansArr.push(gameArr[i]);

    return ansArr;

}

const { request } = require("express");

const express = require("express");

const app = express();

const port = 5000;

let server = app.listen(port);

console.log(`Server on port ${port}`);

app.set("view engine", "hbs");

const way = \_\_dirname + "/static";

app.use(express.static(way));

fillGameArr();

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();

});

// выдача страницы с массивом учеников

app.get("/show\_games", function(request, response) {

    const age = request.query.age;

    let info = {

        ageValue: age,

        gameArr: getGames(age)

    }

    response.render("pageGames.hbs", info);

});

**pageGames.hbs**

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Подборка игр</title>

</head>

<body>

<h2>

    Список игр для возраста <i>{{ageValue}}</i>:

</h2>

{{#each gameArr}}

    <div style="background: rgb(232, 240, 254); margin-bottom: 10px; padding: 5px; font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;">

        <h3>{{this.name}} (<i>{{this.age\_limit}}+</i>)</h3>

        <div style="background: rgb(134, 188, 238); margin-bottom: 3px; padding: 7px; font-family: Verdana, Geneva, Tahoma, sans-serif;">

            Описание: {{this.description}}

        </div>

    </div>

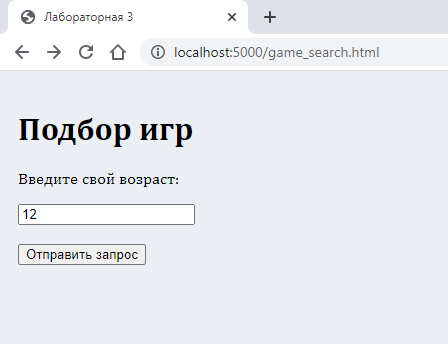
{{/each}}

</body>

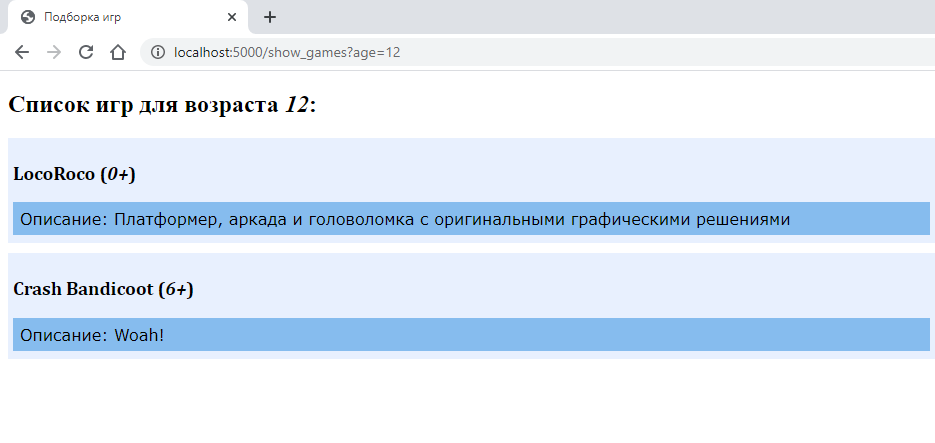
</html>

**Тесты**

Ввод:



Вывод:



Задание 6.2

Создать сервер. В оперативной памяти на стороне сервера создать массив, в котором хранится информация о пользователях (логин, пароль, хобби, возраст). На основе **cookie** реализовать авторизацию пользователей. Реализовать возможность для авторизованного пользователя просматривать информацию о себе.

**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");

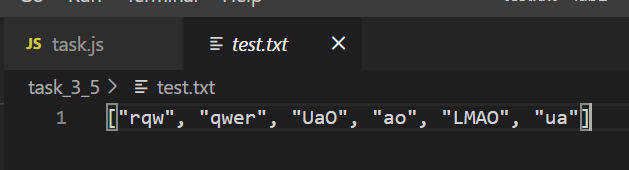
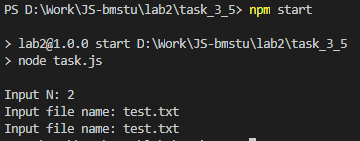
    }

}

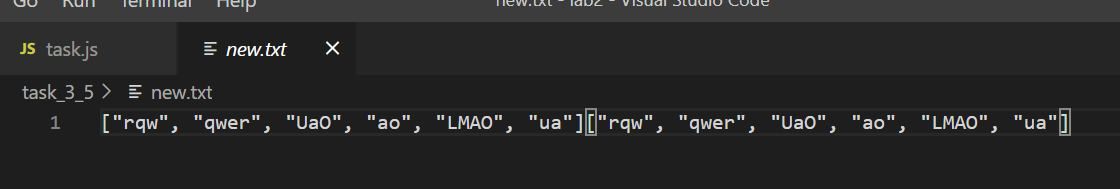
fs.writeFileSync("new.txt", result\_str);

**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;

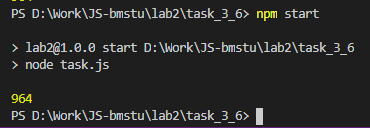
        step /= 2;

    }

}

console.log(size);

**Тесты**



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);

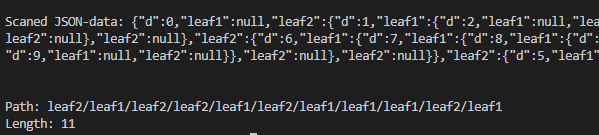
**Тесты**

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":null}},"leaf2":{"d":5,"leaf1":null,"leaf2":null}}}},"leaf2":null}}

Вывод:



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";

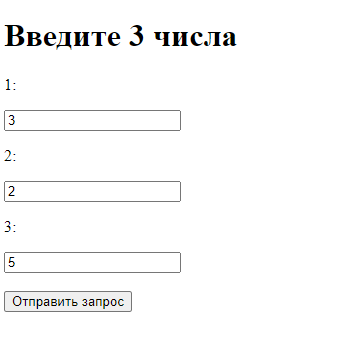
    }

    response.end(answer);

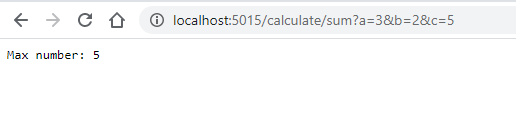
});

**Тесты**

Ввод:



Вывод:



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";

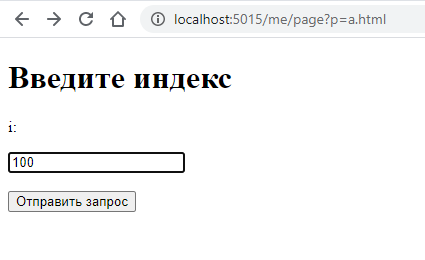
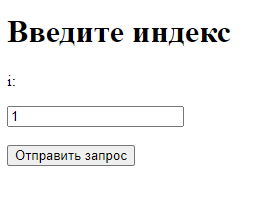
    }

    response.end(answer);

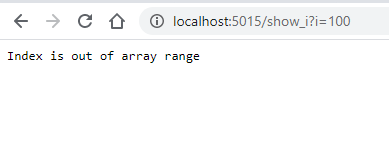
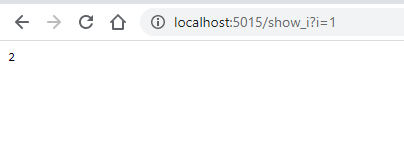
});

**Тесты**

Ввод:



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 addres: ")

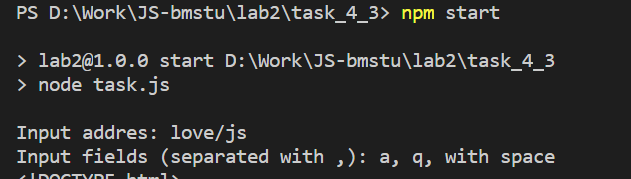
const field\_arr = readlineSync.question("Input fields (separated with ,): ").split(", ");

generate\_html(req\_addr, field\_arr);

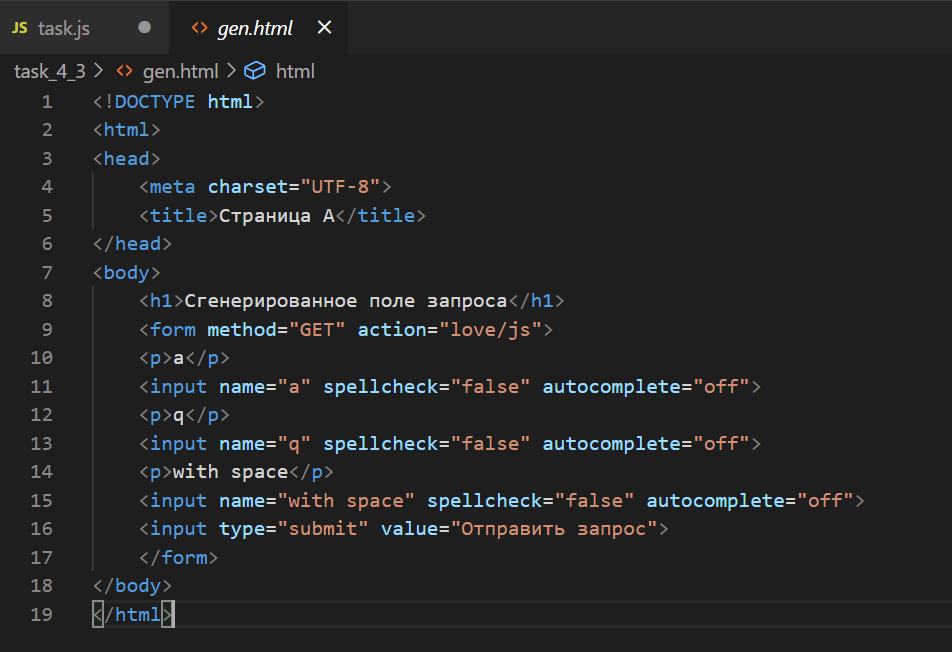
show\_html();

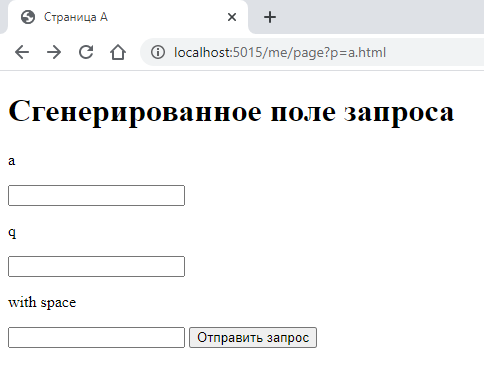
**Тесты**

Ввод:



Вывод:





Задание 4.4

Запустить сервер. Реализовать на сервере функцию, которая принимает на вход числа A, B и C. Функция должна выдавать массив целых чисел на отрезке от A до B, которые делятся на C 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);

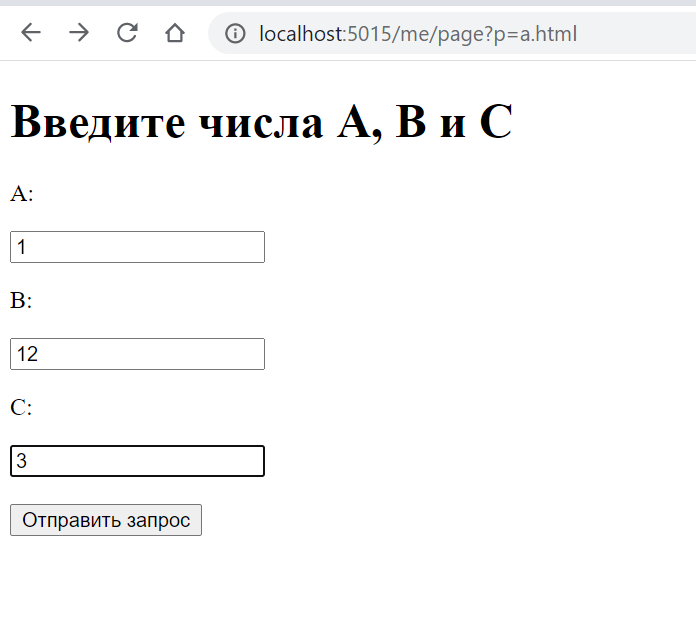
    }

    response.end(answer);

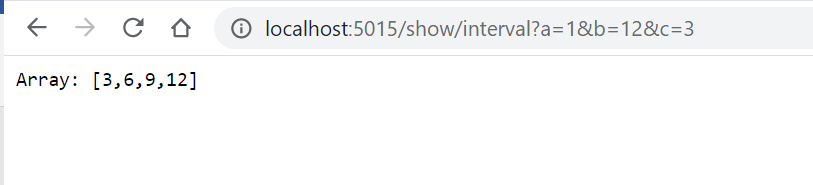
});

**Тесты**

Ввод:



Вывод:



**Вывод**

В рамках лабораторной работы было выполнено ознакомление и практическое закрепление основ работы с AJAX запросами GET и POST в языке JS.