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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**
НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

О Т Ч Е Т

По лабораторной работе № 3

Дисциплина: Архитектура ЭВМ

Студент ИУ7-52Б _____ Н.А. Гарасев
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ А.Ю. Попов
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Цель работы:

Приобретение и закрепление на практике следующих навыков:

1. Получение статических файлов

2. AJAX запросы GET

3. POST запросы в AJAX

4. Работа с шаблонизатором

5. Сессии в NodeJS

Task 5

1. Задание 1

Условие задачи:

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.

В условие задачи не сказано, в каком формате хранятся данные в файле,

поэтому я храню их в следующем виде:

[{"mail":"1bk@mail.ru","surname":"Garasev","phone":"7655"},
{"mail":"2bk@mail.ru","surname":"Kulicov","phone":"3455"},
{"mail":"nikita","surname":"da","phone":"ya"},
{"mail":"garas","surname":"nikita","phone":"915"}]

Листинг:

index.js

```
"use strict";


// импортируем библиотеку
const express = require("express");
const fs = require("fs");
```

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

// отправка статических файлов
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 = [];
    request.on('data', (chunk) => {
        body.push(chunk);
    }).on('end', () => {
        body = Buffer.concat(body).toString();
        callback(body);
    });
}

function checkUniqe(obj, mail, phone) {
    for (let i = 0; i < obj.length; i++) {
        if (obj[i].mail === mail || obj[i].phone === phone) {
```

```
            return false;
        }
    }
    return true;
}

// it is post
app.post("/save/info", function(request, response) {
    loadBody(request, function(body) {
        const obj = JSON.parse(body);
        const mail = obj["mail"];
        const surname = obj["surname"];
        const phone = obj["phone"];

        let contentString = fs.readFileSync("./file.txt", "utf8");
        let answerString = "Не добавили человека";
        if (mail === '' || surname === '' || phone === '') {
            answerString = "Заполните поля";
        } else {
            let obj;
            if (contentString === '') {
                obj = [];
            } else {
                obj = JSON.parse(contentString);
            }
            if (checkUniqe(obj, mail, phone)) {
                obj.push({"mail": mail, "surname": surname, "phone":phone})
                contentString = JSON.stringify(obj);
                fs.writeFileSync("./file.txt", contentString);
                answerString = "Добавили человека"
            }
        }
        response.end(JSON.stringify({
```

```
            answer: answerString,
            curFile: contentString
        }));
    });
});
```

static/page.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Моя страница</title>
    <link rel="stylesheet" href="/style.css">
</head>
<body>
    <p>Почта</p>
    <input id="field-mail" type="text" spellcheck="false" autocomplete="off">

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

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

    <br>
    <br>

    <div id="btn-send" class="btn-class" onclick="makeAction()">Отправить</div>

    <br>
    <br>
```

```
    <h1 id="result-label"></h1>

    <script src="/code.js"></script>
</body>
</html>
```

static/code.js

```
"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);
    }
}

function makeAction() {
    // input fields
    const mail = document.getElementById("field-mail").value;
    const surname = document.getElementById("field-surname").value;
    const phone = document.getElementById("field-phone").value;

    // label
    const label = document.getElementById("result-label");

    ajaxPost("/save/info", JSON.stringify({
        mail, surname, phone
```

```
    }), function(answerString) {
        const answerObject = JSON.parse(answerString);
        const answer = answerObject.answer;
        alert(answer);
        const curFile = answerObject.curFile;
        label.innerHTML = `Текущий файл: ${curFile}`;
    });
}
```

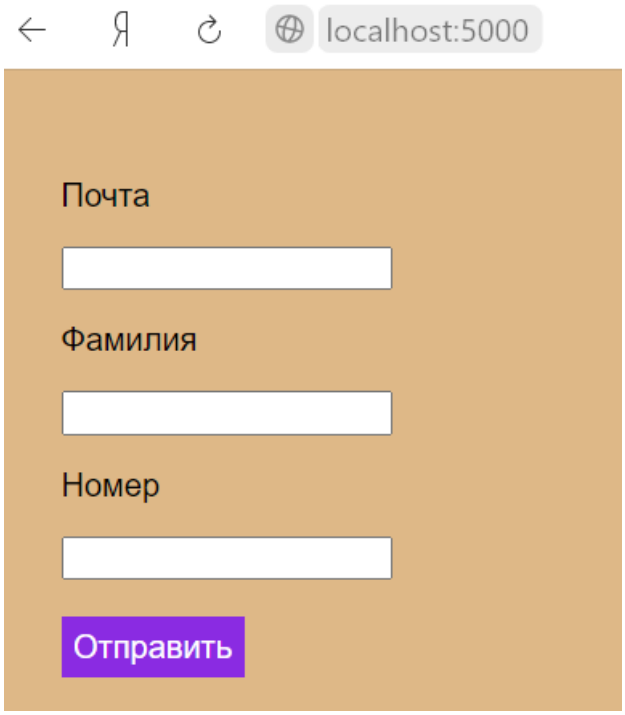
static/style.css

```
body {
    padding: 30px;
    background: burlywood;
    font-family: Geneva, Arial, Helvetica, sans-serif;
}

.btn-class {
    padding: 6px;
    background: blueviolet;
    color: white;
    cursor: pointer;
    display: inline-block;
}
```

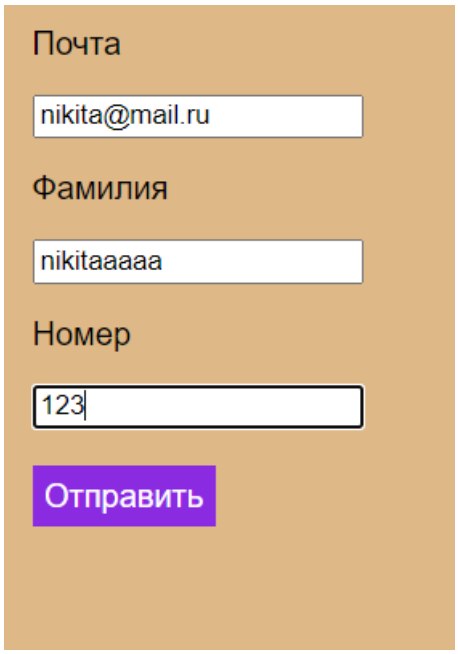
Тестирование:

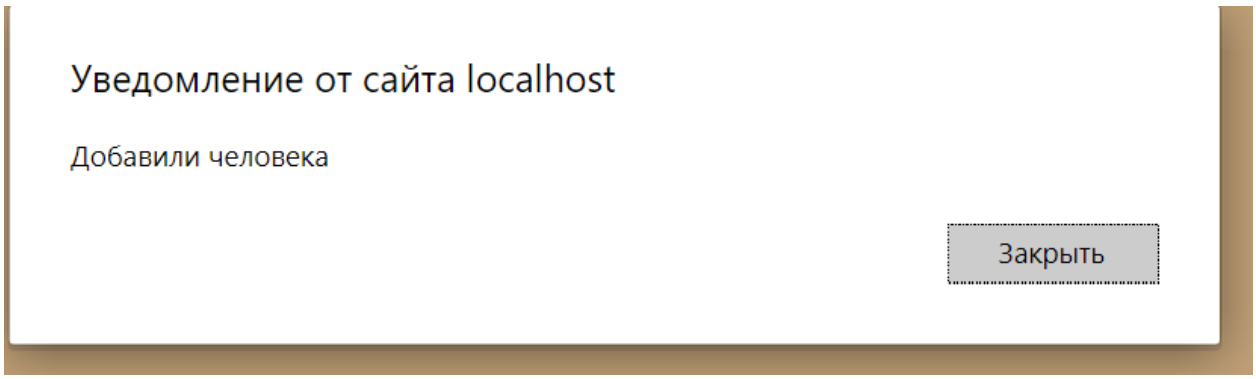
Изначально файл, куда записываем данные - пуст.

Отправим пустые поля



Добавим человека

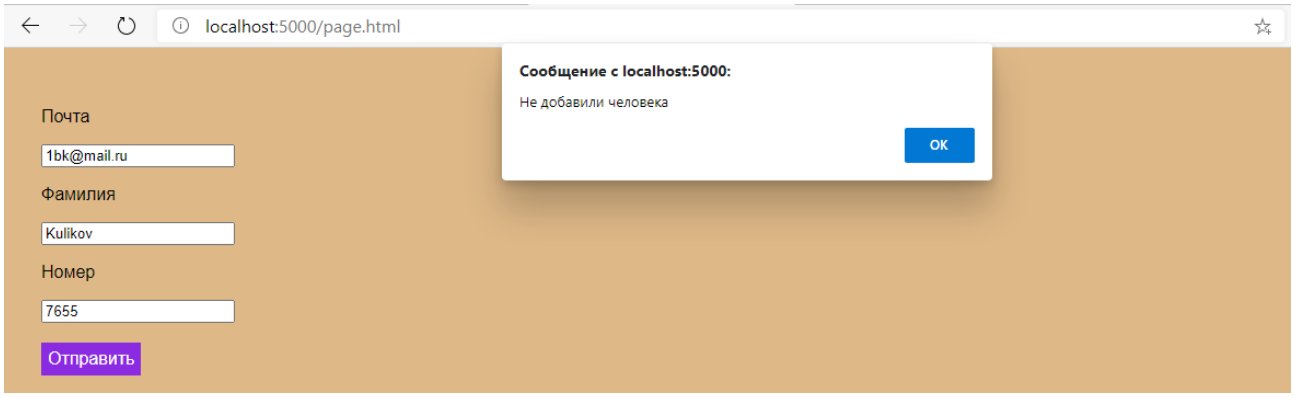




Содержимое файла изменилось.

**Текущий файл: [{"mail":"1bk@mail.ru","surname":"Garasev","phone":"7655"},
{"mail":"2bk@mail.ru","surname":"Kulicov","phone":"3455"},
{"mail":"nikita","surname":"da","phone":"ya"},
{"mail":"garas","surname":"nikita","phone":"915"},
{"mail":"nikita@mail.ru","surname":"nikitaaaaa","phone":"123"}]**

Добавим этого же человека еще раз.



Содержимое файла не изменилось.

2. Задание 2

Условие задачи:

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.

Листинг:

В index.js добавим строчки кода

```
// выдать страницу
app.get("/page", function(request, response) {
    response.sendFile(__dirname + "/" + "page2.html");
});

function findObj(key, obj) {
    for (let i = 0; i < obj.length; i++) {
        if (obj[i].mail === key) {
            return obj[i];
        }
    }
    return null;
}

// выдать запись
app.get("/record", function(request, response) {
    const key = request.query.k;
    let value;
    let contentString = fs.readFileSync("./file.txt", "utf8");
```

```
    if (key === '') {
        value = 'Вы не заполняли поле почты'
    } else {
        if (contentString === '') {
            value = 'Нет людей в файле'
        } else {
            const obj = JSON.parse(contentString);
            let fobj = findObj(key, obj);
            if (fobj != null) {
                value = `mail: ${fobj.mail}, surname: ${fobj.surname}, phone: ${f
obj.phone}`
            } else {
                value = "Нет такого человека"
            }
        }
    }
    response.end(JSON.stringify({
        v: value
    }));
});
```

static/page2.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Страница</title>
</head>
<body>

<p>Почта</p>
<input id="field-mail" type="text" spellcheck="false" autocomplete="off">
```

```
<br>
<br>
<button onclick="getRecord()">Отправить запрос</button>

<h1 id="result-label"></h1>

<script src="/code2.js"></script>

</body>
</html>
```

static/code2.js

```
"use strict";

// ajax get
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);
    };
};

    // ввод ключа и отправка запроса на сервер
function getRecord() {
    // input fields
    const mail = document.getElementById("field-mail").value;

    const mail_a = encodeURIComponent(mail);
```

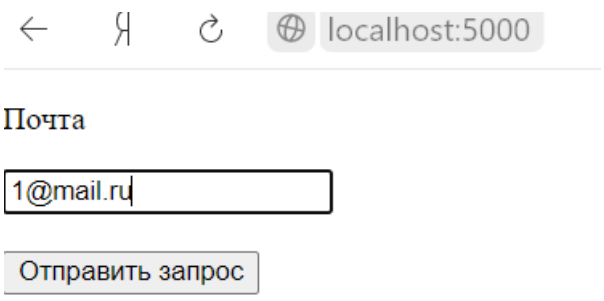
```
    const url = `/record?k=${mail_a}`;

    // label
    const label = document.getElementById("result-label");

    // отправка запроса на сервер
    ajaxGet(url, function(answerString) {
        const answerObject = JSON.parse(answerString);
        const result = answerObject.v;
        label.innerHTML = `Ответ: ${result}`;
    });
}
```

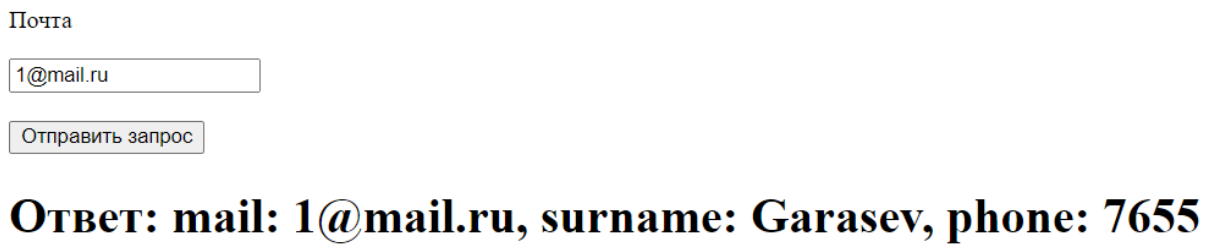
Тестирование:

В файл хранится следующая информация

[{"mail":"1@mail.ru","surname":"Garasev","phone":"7655"},

{"mail":"2@mail.ru","surname":"Dkul","phone":"3455"}]





3. Задание 3

Условие задачи:

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.

Листинг:

style1.css

```css
body {
    padding: 30px;
    background: tomato;
    font-size: 16px;
    line-height: 32px;
}

p {
    font-family: Courier New;
    font-weight: bold;
}

h1 {
    font-size: 40px;
    color: #333333;
    font-weight: 500;

    margin-top: 50px;
    margin-left: 40px;
    margin-bottom: 30px;
    margin-right: 0px;
}
```

```css
.btn-class {
    width: 160px;
    height: 70px;

    font-size: 16px;
    font-weight: 600;
    color: white;
    background-color: #cc9933;
    text-align: center;
    font-family: sans-serif;

    border: solid 1px #cc9933;
    border-radius: 12px;

    cursor: pointer;
}



body {
    padding: 30px;
    background: pink;
    font-size: 16px;
    line-height: 32px;
}

p {
    color: #ffcc66;
}

h1 {
    font-size: 40px;
    color: #333333;
    font-weight: 500;

    margin-top: 50px;
    margin-left: 40px;
    margin-bottom: 30px;
    margin-right: 0px;
}

.btn-class {
    width: 160px;
    height: 70px;

    font-size: 16px;
    font-weight: 600;
    color: white;
    background-color: #cc9933;
    text-align: center;
    font-family: sans-serif;

    border: solid 1px #cc9933;
    border-radius: 12px;
```

```
    cursor: pointer;
}
```

Тестирование:

page1.html

Task 6

1. Задание 1

Условие задачи:

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.

Листинг:

index.js

```
"use strict";


// импорт библиотеки
const express = require("express");


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


// активируем шаблонизатор
```

```
app.set("view engine", "hbs");


// 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();
});

let gameArray = [
    {name: "Divinity", description: "RPG", age: 18},
    {name: "gta", description: "action", age: 18},
    {name: "cs-go", description: "shooter", age: 12},
    {name: "minecraft", description: "game for child", age: 6}
];

// выдача страницы с играми
app.get("/page/games", function(request, response) {
    const key = request.query.k;
    console.log(key);
    const infoObject = {
        descriptionValue: `Список компьютерных игр, с возрастным ограничением ${k
ey}`,
        games : gameArray.filter(obj => obj.age <= key)
    };
    response.render("pageGames.hbs", infoObject);
});
```

pageGames.hbs

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="UTF-8">
    <title>Компьютерные игры</title>
</head>
<body>

<h2>
    {{descriptionValue}}
</h2>

{{#each games}}
    <div style="background: yellow; margin-bottom: 15px; padding: 8px;">
        Название: {{this.name}}
        <br>
        Описание: {{this.description}}
        <br>
        Возраст: {{this.age}}
    </div>
{{/each}}

</body>
</html>
```

Тестирование:

Список компьютерных игр, с возрастным ограничением 10

Название: minecraft
Описание: game for child
Возраст: 6

Список компьютерных игр, с возрастным ограничением 20

Название: Divinity
Описание: RPG
Возраст: 18

Название: gta
Описание: action
Возраст: 18

Название: cs-go
Описание: shooter
Возраст: 12

Название: minecraft
Описание: game for child
Возраст: 6

2.Задание 2

Условие задачи:

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.

Листинг:

index.js

```
"use strict";


// импортируем библиотеки
const express = require("express");
```

```
const cookieSession = require("cookie-session");
const fs = require("fs");

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

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

const users = [
    {"login" : "nikita", "password" : "1830", "age" : 20, "hobby" : "tennis"},
    {"login" : "dima", "password" : "da", "age" : 1, "hobby" : "peniboards"}
]

// работа с сессией
app.use(cookieSession({
    name: 'session',
    keys: ['hhh', 'qqq', 'vvv'],
    maxAge: 24 * 60 * 60 * 1000 * 365
}));

// 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");
    next();
});

// сохранить cookie
```

```
app.get("/request", function(request, response) {
    // получаем параметры запроса
    const login = request.query.login;
    const password = request.query.password;



    let flagAuth = false;
    let answerString = "false";
    for (let i = 0; i < users.length; i++) {
        if (users[i].login === login && users[i].password === password) {
            flagAuth = true;
            break;
        }
    }
    if (flagAuth) {
        request.session.login = login;
        request.session.password = password;
        answerString = "true";
    }
    response.end(JSON.stringify({"answer" : answerString}));
});

app.get("/info", function(request, response) {
    // получаем параметры запроса
    const login = request.session.login
    const password = request.session.password

    for (let i = 0; i < users.length; i++) {
        if (users[i].login === login && users[i].password === password)
            response.end(JSON.stringify(users[i]))
    }
});
```

```
// получить cookie
app.get("/api/get", function(request, response) {
    // контролируем существование cookie
    if(!request.session.login) return response.end("Not exists");
    if(!request.session.age) return response.end("Not exists");
    // отправляем ответ с содержимым cookie
    const login = request.session.login;
    const age = request.session.age;
    response.end(JSON.stringify({
        login,
        age
    }));
});

// удалить все cookie
app.get("/api/delete", function(request, response) {
    request.session = null;
    response.end("Delete cookie ok");
});
```

auth.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Авторизация</title>
    <link rel="stylesheet" href="/style.css">
</head>
<body>
    <p>Логин</p>
    <input id="field-login" type="text" spellcheck="false" autocomplete="off">
```

```
    <p>Пароль</p>
    <input id="field-password" type="text" spellcheck="false" autocomplete="off">

    <br>
    <br>

    <div id="btn-send" class="btn-class" onclick="makeAction()">Войти</div>
    <script src="/codeAuth.js"></script>
</body>
</html>
```

codeAuth.js

```
"use strict";

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

function makeAction() {
    const login = document.getElementById("field-login");
    const password = document.getElementById("field-password");

    const l = login.value;
    const p = password.value;
    const url = `request?login=${l}&password=${p}`
    ajaxGet(url, function(answerString) {
        const answerObject = JSON.parse(answerString);
```

```
        alert(answerObject.answer);
        if (answerObject.answer === "true")
            window.location.replace("http://localhost:5000/profile.html")
        else
            alert("Not auth user");
    });
}
```

profile.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Профиль</title>
    <link rel="stylesheet" href="/style.css">
</head>
<body>

    <div id="btn-send" class="btn-
class" onclick="makeAction()">Получить данные</div>

    <br>
    <br>

    <h1 id="result-label"></h1>

    <script src="/codeProfile.js"></script>
</body>
</html>
```

codeProfile.js

```
function ajaxGet(urlString, callback) {
```

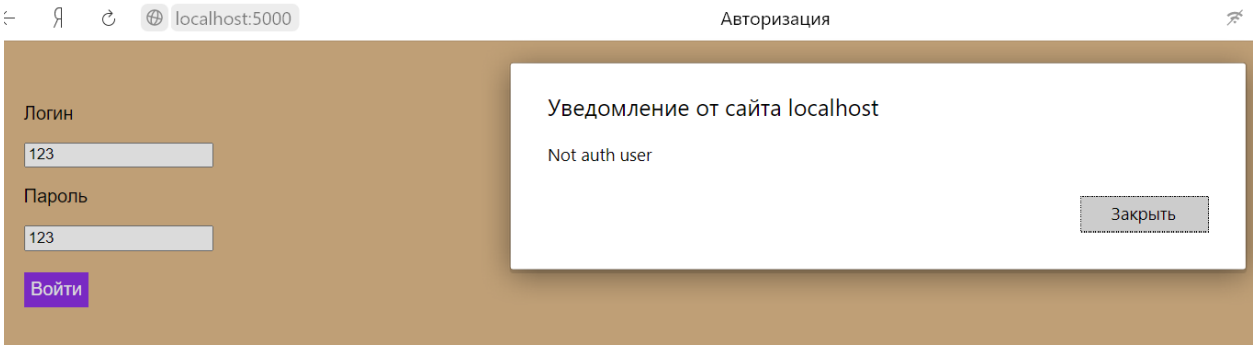
```
    let r = new XMLHttpRequest();
    r.open("GET", urlString, true);
    r.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
    r.send(null);
    r.onload = function() {
        callback(r.response);
    }
}

function makeAction() {
    const label = document.getElementById("result-label");
    const url = `/info`
    ajaxGet(url, function(answerString) {
        const obj = JSON.parse(answerString);
        label.innerHTML = `Логин: ${obj.login} Возраст: ${obj.age} Хобби: ${obj.h
obby}`
    });
}
```

style.css

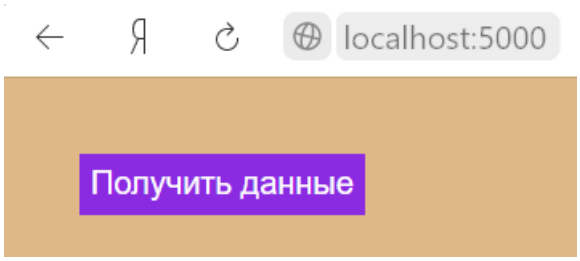
```
body {
    padding: 30px;
    background: burlywood;
    font-family: Geneva, Arial, Helvetica, sans-serif;
}

.btn-class {
    padding: 6px;
    background: blueviolet;
    color: white;
    cursor: pointer;
    display: inline-block;
}
```

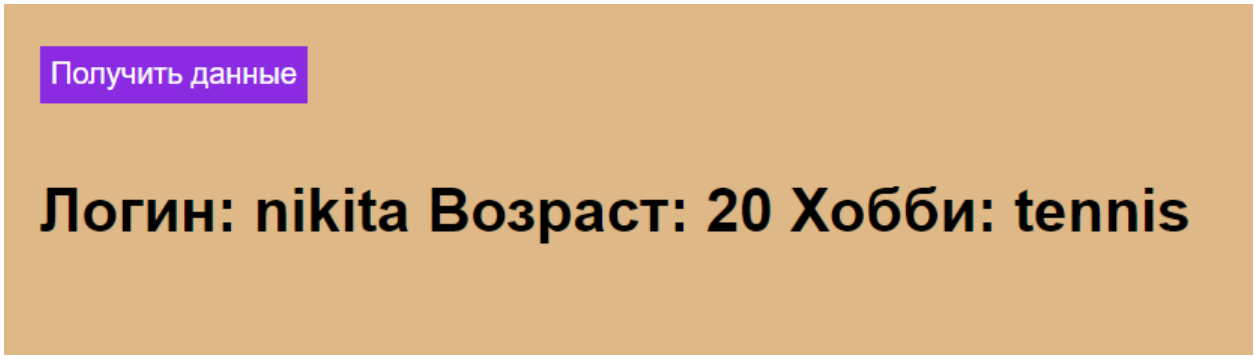
Тестирование:

Нет такого пользователя



Авторизованный пользователь





Вывод:

Все поставленные задачи были выполнены. Цель лабораторной работы достигнута.