|  |  |
| --- | --- |
| 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 №** | 4 |

**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 взаимодействием серверов, скриптами и дочерними процессами.

Задание 7.1

Создать сервер **А**. На стороне сервера хранится файл с содержимым в формате **JSON**. При получении запроса на **/insert/record** идёт добавление записи в файл. При получении запроса на **/select/record** идёт получение записи из файла. Каждая запись хранит информацию о машине (*название* и *стоимость*).

Создать сервер **Б**. На стороне сервера хранится файл с содержимым в формате **JSON**. Каждая запись в файле хранит информацию о складе и массиве машин, находящихся на данном складе. То есть каждая запись хранит в себе название склада (*строку*) и массив названий машин (*массив строк*). При получении запроса на **/insert/record** идёт добавление записи в файл. При получении запроса на **/select/record** идёт получение записи из файла.

Создать сервер **C**. Сервер выдаёт пользователю страницы с формами для ввода информации. При этом сервер взаимодействует с серверами **А** и **Б**. Реализовать для пользователя функции:

* создание нового типа машины
* получение информации о стоимости машины по её типу
* создание нового склада с находящимися в нём машинами
* получение информации о машинах на складе по названию склада

Реализовать удобный для пользователя интерфейс взаимодействия с системой (использовать поля ввода и кнопки).

**Программная реализация**

**servA.js (сервер A)**

"use strict";

const express = require("express");

const fs = require("fs");

const app = express();

const port = 5003;

app.listen(port);

console.log("Car server on port " + port);

const file\_name = "A.txt";

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();

});

function get\_file\_arr(f\_name) {

    let file\_arr;

    if (fs.existsSync(f\_name)) {

        const file\_str = fs.readFileSync(f\_name, "utf-8");

        if (file\_str == "")

            file\_arr = [];

        else

            file\_arr = JSON.parse(file\_str);

    } else {

        file\_arr = [];

    }

    return file\_arr;

}

function is\_valid(car) {

    if (!car.name)  return false;

    if (!car.cost)  return false;

    return true;

}

function is\_car\_in(car\_arr, car) {

    for (let i=0; i<car\_arr.length; i++)

        if (car\_arr[i].name == car.name)

            return true;

    return false;

}

function find\_car(car\_arr, name) {

    for (let i=0; i<car\_arr.length; i++)

        if (car\_arr[i].name == name)

            return car\_arr[i];

    return null;

}

function loadBody(request, callback) {

    let body = [];

    request.on('data', (chunk) => {

        body.push(chunk);

    }).on('end', () => {

        body = Buffer.concat(body).toString();

        callback(body);

    });

}

app.post("/insert/record", function(request, response) {

    loadBody(request, function(body) {

        let msg = "Car added";

        const obj = JSON.parse(body);

        let file\_arr = get\_file\_arr(file\_name);

        if (!is\_car\_in(file\_arr, obj)) {

            file\_arr.push(obj);

            fs.writeFileSync(file\_name, JSON.stringify(file\_arr));

        } else {

            msg = "Car allready in file";

        }

        response.end(msg);

    });

});

app.post("/select/record", function(request, response) {

    loadBody(request, function(body) {

        const car\_name = JSON.parse(body);

        const file\_arr = get\_file\_arr(file\_name);

        let car = find\_car(file\_arr, car\_name);

        if (car === null)

            response.end();

        else

            response.end(JSON.stringify(car));

    });

});

app.post("/select/all", function(request, response) {

    loadBody(request, function(body) {

        const file\_arr = get\_file\_arr(file\_name);

        response.end(JSON.stringify(file\_arr));

    });

});

**servB.js (сервер B)**

"use strict";

const express = require("express");

const fs = require("fs");

const app = express();

const port = 5002;

app.listen(port);

console.log("Storage server on port " + port);

const file\_name = "B.txt";

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();

});

function get\_file\_arr(f\_name) {

    let file\_arr;

    if (fs.existsSync(f\_name)) {

        const file\_str = fs.readFileSync(f\_name, "utf-8");

        if (file\_str == "")

            file\_arr = [];

        else

            file\_arr = JSON.parse(file\_str);

    } else {

        file\_arr = [];

    }

    return file\_arr;

}

function is\_valid(storage) {

    if (!storage.name)  return false;

    if (!storage.car\_arr)  return false;

    return true;

}

function is\_storage\_in(storage\_arr, storage) {

    for (let i=0; i<storage\_arr.length; i++)

        if (storage\_arr[i].name == storage.name)

            return true;

    return false;

}

function find\_storage(storage\_arr, name) {

    for (let i=0; i<storage\_arr.length; i++)

        if (storage\_arr[i].name == name)

            return storage\_arr[i];

    return null;

}

function loadBody(request, callback) {

    let body = [];

    request.on('data', (chunk) => {

        body.push(chunk);

    }).on('end', () => {

        body = Buffer.concat(body).toString();

        callback(body);

    });

}

app.post("/insert/record", function(request, response) {

    loadBody(request, function(body) {

        let msg = "Storage added";

        const obj = JSON.parse(body);

        let file\_arr = get\_file\_arr(file\_name);

        if (!is\_storage\_in(file\_arr, obj)) {

            file\_arr.push(obj);

            fs.writeFileSync(file\_name, JSON.stringify(file\_arr));

        } else {

            msg = "Storage allready in file";

        }

        response.end(msg);

    });

});

app.post("/select/record", function(request, response) {

    loadBody(request, function(body) {

        const storage\_name = JSON.parse(body);

        const file\_arr = get\_file\_arr(file\_name);

        let storage = find\_storage(file\_arr, storage\_name);

        if (storage === null)

            response.end();

        else

            response.end(JSON.stringify(storage));

    });

});

**servC.js (сервер C)**

"use strict";

const express = require("express");

const request = require("request");

const app = express();

const port = 5000;

app.listen(port);

console.log(`Web-interface server on port ${port}`);

const way = \_\_dirname + "/static";

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

app.set("view engine", "hbs");

const car\_addr = "http://localhost:5003";

const storage\_addr = "http://localhost:5002";

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();

});

function sendPost(url, body, callback) {

    const headers = {};

    headers["Cache-Control"] = "no-cache, no-store, must-revalidate";

    headers["Connection"] = "close";

    request.post({

        url: url,

        body: body,

        headers: headers,

    }, function (error, response, body) {

        if(error) {

            callback(null);

        } else {

            callback(body);

        }

    });

}

app.get("/insert/car", function(request, response) {

    const name\_ = request.query.name;

    const cost\_ = request.query.cost;

    sendPost(car\_addr + "/insert/record", JSON.stringify({

        name: name\_,

        cost: cost\_

    }), function(answerString) {

        response.end("Result: " + answerString);

    });

});

app.get("/insert/storage", function(request, response) {

    const name\_ = request.query.name;

    const car\_arr\_ = request.query.car\_arr;

    sendPost(storage\_addr + "/insert/record", JSON.stringify({

        name: name\_,

        car\_arr: car\_arr\_

    }), function(answerString) {

        response.end("Result: " + answerString);

    });

});

app.get("/select/car", function(request, response) {

    const name\_ = request.query.name;

    sendPost(car\_addr + "/select/record", JSON.stringify(name\_), function(answerString) {

        response.end(answerString);

    });

});

app.get("/select/storage", function(request, response) {

    const name\_ = request.query.name;

    sendPost(storage\_addr + "/select/record", JSON.stringify(name\_), function(answerString) {

        response.end(answerString);

    });

});

// Jump to other pages

app.get("/show/insert/car", function(request, response) {

    response.render("in\_car.hbs", null);

});

app.get("/show/select/car", function(request, response) {

    response.render("out\_car.hbs", null);

});

app.get("/show/insert/storage", function(request, response) {

    sendPost(car\_addr + "/select/all", "", function(answerString) {

        let obj = { car\_arr: JSON.parse(answerString)}

        response.render("in\_storage.hbs", obj);

    });

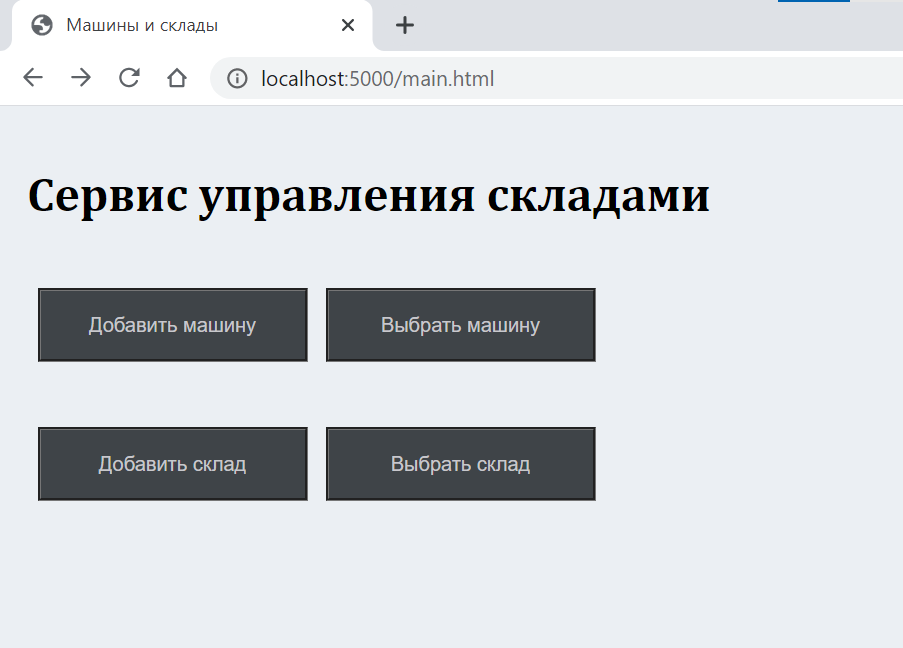
});

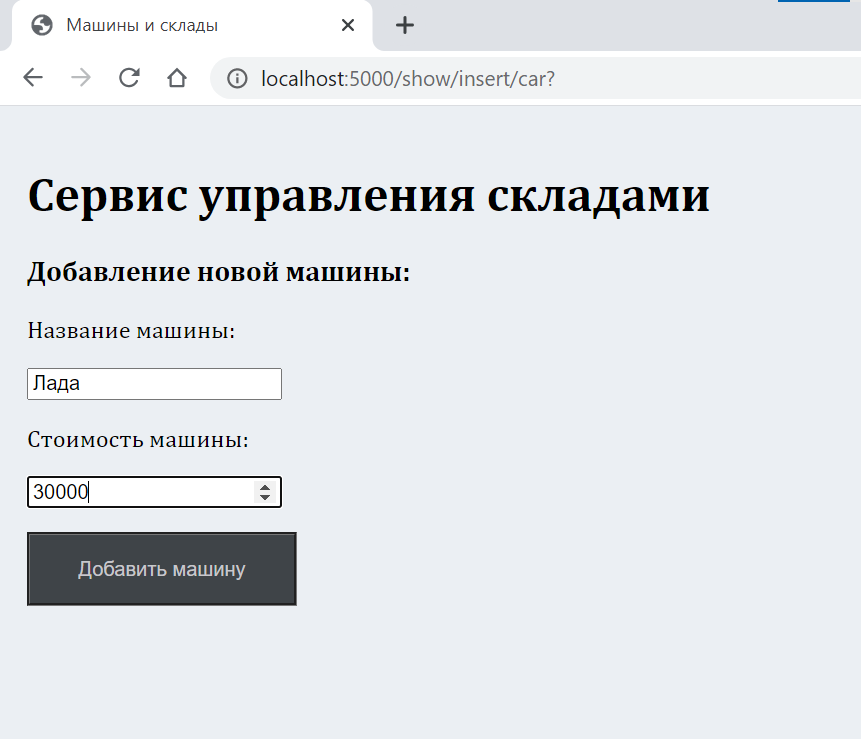
app.get("/show/select/storage", function(request, response) {

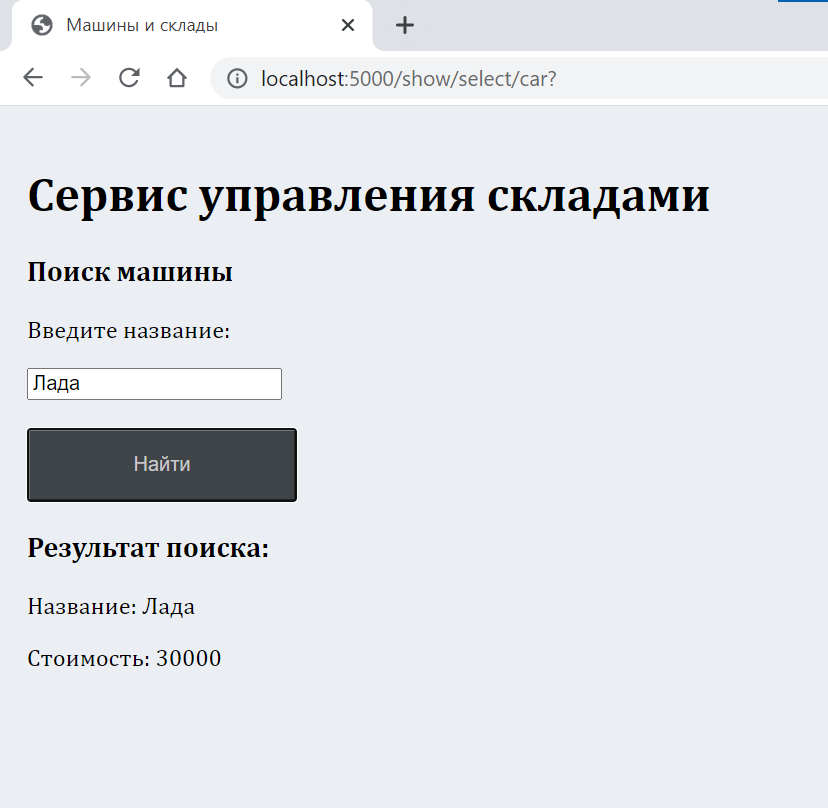
    response.render("out\_storage.hbs", null);

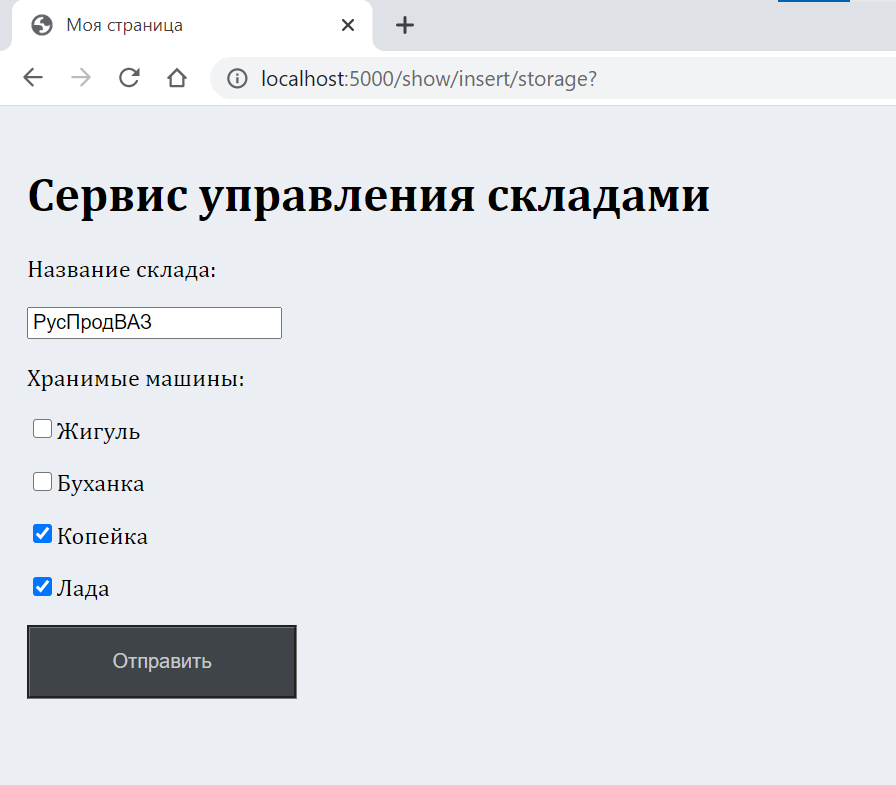
});

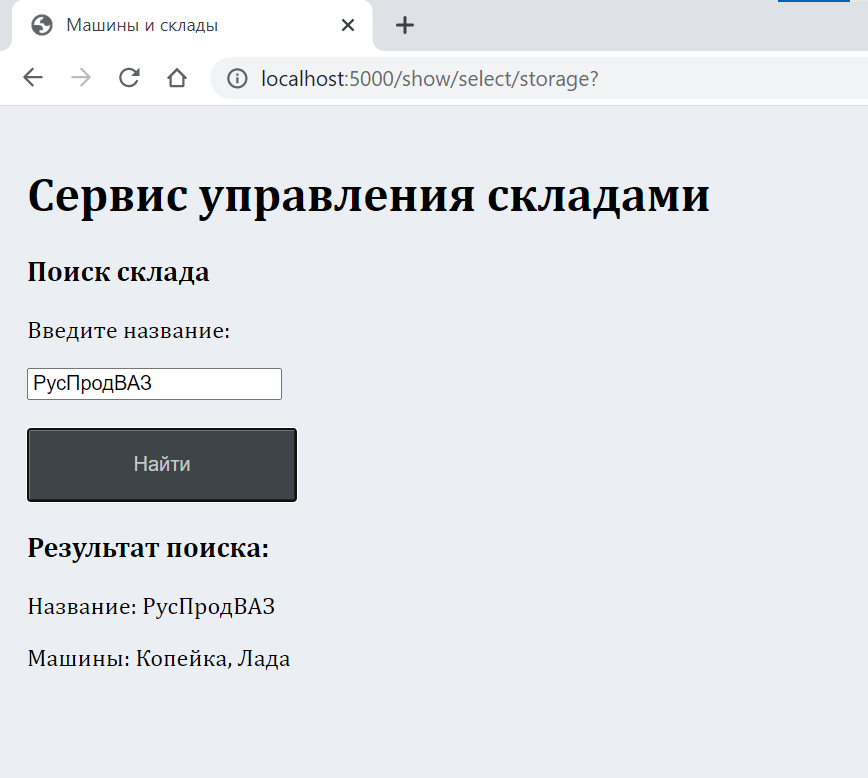
**Тесты**











Задание 7.2

Написать скрипт, который принимает на вход число и считает его факториал. Скрипт должен получать параметр через **process.argv**.

Написать скрипт, который принимает на вход массив чисел и выводит на экран факториал каждого числа из массива. Скрипт принимает параметры через **process.argv**.

При решении задачи вызывать скрипт вычисления факториала через **execSync**.

**Программная реализация**

**factorial.js**

"use strict"

function factroial(n) {

    let ans = 1;

    for (let i=2; i<=n; i++)

        ans \*= i;

    return ans;

}

console.log(factroial(parseInt(process.argv[2])));

**arr\_factorial.js**

"use strict;"

const execute = require("child\_process").execSync;

function get\_factorial(n) {

    const options = {encoding : 'utf8'}

    const command = `node factorial.js ${n}`;

    const ans = execute(command, options);

    return parseInt(ans);

}

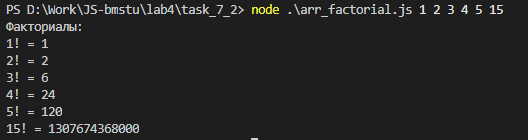
console.log("Факториалы:");

for (let i=2; i<process.argv.length; i++)

    console.log(`${process.argv[i]}! = ${get\_factorial(process.argv[i])}`);

            obj

**Тесты**



**Вывод**

В рамках лабораторной работы было выполнено ознакомление и практическое закрепление основ работы с взаимодействием серверов, скриптами и дочерними процессами.