|  |  |
| --- | --- |
| 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 **09.03.04 Программная инженерия**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 4 |

**Название:**

Работа с несколькими серверами, скриптом, дочерними процессами

**Дисциплина:** Архитектура ЭВМ

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

Москва, 2020

**Цель:** научиться работать с несколькими серверами, передавать параметры скрипту и изучить дочерние процессы и способы взаимодействия между ними.

**Задание 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).

**index\_A.js (Сервер А)**

let file = "cars.txt";

// импорт библиотеки

const express = require("express");

const fs = require("fs");

// запускаем сервер

const app = express();

const port = 5002;

app.listen(port);

console.log("Server on port " + port);

// заголовки для ответа

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();

});

// загрузка тела

function loadBody(request, callback) {

    let body = [];

    request.on('data', (chunk) => {

        body.push(chunk);

    }).on('end', () => {

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

        callback(body);

    });

}

function add\_record(name, price) {

    let data = [];

    data.push(fs.readFileSync(file, "utf-8"));

    if (data != '')

    {

        let temp = JSON.parse(data);

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

            if (temp[i].name == name)

                return "ERROR: car is already exist";

        temp.push({"name": name, "price": price});

        data = temp;

    }

    else

        data[0] = {"name": name, "price": price};

    fs.writeFileSync(file, JSON.stringify(data))

    return "Information was added";

}

function show\_record(name)

{

    let data = [];

    data.push(fs.readFileSync(file, "utf-8"));

    if (data != '')

    {

        let temp = JSON.parse(data);

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

            if (temp[i].name == name)

                return temp[i];

    }

    return {"name": null, "price": null};

}

// приём запроса

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

    loadBody(request, function(body) {

        const obj = JSON.parse(body);

        const name = obj["name"];

        const price = obj["price"];

        let msg = add\_record(name, price);

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

    });

});

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

    loadBody(request, function(body) {

        const obj = JSON.parse(body);

        const name = obj["name"];

        let car\_info = show\_record(name);

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

    });

});

**index\_B.js (Сервер B)**

let file = "stores.txt";

// импорт библиотек

const express = require("express");

const fs = require("fs");

// запускаем сервер

const app = express();

const port = 5003;

app.listen(port);

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

// 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();

});

// загрузка тела

function loadBody(request, callback) {

    let body = [];

    request.on('data', (chunk) => {

        body.push(chunk);

    }).on('end', () => {

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

        callback(body);

    });

}

function add\_record(name, cars) {

    let data = [];

    data.push(fs.readFileSync(file, "utf-8"));

    if (data != '')

    {

        let temp = JSON.parse(data);

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

            if (temp[i].name == name)

                return "ERROR: store is already exist";

        temp.push({"name": name, "cars": cars});

        data = temp;

    }

    else

        data[0] = {"name": name, "cars": cars};

    fs.writeFileSync(file, JSON.stringify(data))

    return "Information was added";

}

function show\_record(name)

{

    let data = [];

    data.push(fs.readFileSync(file, "utf-8"));

    if (data != '')

    {

        let temp = JSON.parse(data);

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

            if (temp[i].name == name)

                return temp[i];

    }

    return {"name": null, "price": null};

}

// приём запроса

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

    loadBody(request, function(body) {

        const obj = JSON.parse(body);

        const name = obj["name"];

        const cars = obj["cars"];

        let msg = add\_record(name, cars);

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

    });

});

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

    loadBody(request, function(body) {

        const obj = JSON.parse(body);

        const name = obj["name"];

        let store\_info = show\_record(name);

        response.end(JSON.stringify(store\_info));

    });

});

**index\_C.js (Сервер C)**

const express = require("express");

const request = require("request");

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));

app.set("view engine", "hbs");

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();

});

// функция для отправки POST запроса на другой сервер

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 price = request.query.price;

    sendPost("http://localhost:5002/insert/record", JSON.stringify({

        name: name,

        price: price

    }), function(answerString) {

        response.end(answerString);

    });

})

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

    const name = request.query.name;

    sendPost("http://localhost:5002/select/record", JSON.stringify({

        name: name,

    }), function(answerString) {

        response.end(answerString);

    });

})

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

    const name = request.query.name;

    const cars = request.query.cars.split(" ");

sendPost("http://localhost:5003/insert/record", JSON.stringify({

        name: name,

        cars: cars

    }), function(answerString) {

        response.end(answerString);

    });

})

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

    const name = request.query.name;

    sendPost("http://localhost:5003/select/record", JSON.stringify({

        name: name,

    }), function(answerString) {

        response.end(answerString);

    });

})

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

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

})

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

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

})

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

    response.render("page\_add\_store.hbs", null);

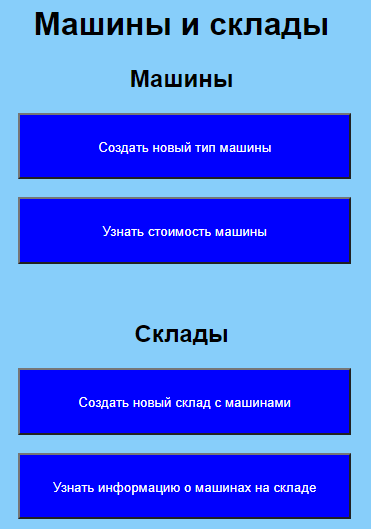
})

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

    response.render("page\_show\_store.hbs", null);

})

**Тесты**

****

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Задание 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.

**index.js**

"use strict";

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

function find\_factorial(number) {

    const options = {encoding: 'utf8'};

    const cmd = `node find\_factorial.js ${number}`;

    const answer = execSync(cmd, options);

    return parseInt(answer);

}

console.log("Число          Факториал");

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

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

**find\_factorial.js**

"use strict";

function factorial(num) {

    if (num < 0)

        return;

    let temp = 1;

    for (let i = 2; i <= num; i++)

        temp \*= i;

    return temp;

}

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

**Тесты:**

|  |  |
| --- | --- |
| Тест | Результат |
|  |  |
|  |  |

**Вывод**

В ходе лабораторной работы была освоена работа с несколькими серверами, скриптами и дочерними процессами. На основе полученных знаний были выполнены соответствующие практические задания и составлен отчет о проделанной работе.