|  |  |
| --- | --- |
| 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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 4 |

**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:**

**Цель работы:**

Приобретение и закрепление на практике следующих навыков:

1. Взаимодействие между серверами. Сервер для отправки запросов на другой сервер.

2. Передача параметров скрипту.

3. Дочерние процессы.

4. Знакомство с Prolog.

### 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).

**Листинг:**

**index.js**

"use strict";

class ServerA {

    static fs = require("fs");

    static express = require("express");

    constructor(port) {

        this.app = ServerA.express();

        this.port = port;

        try {

            this.app.listen(this.port);

            console.log(` Starting server on port ${this.port}... `);

        } catch (error) {

            console.log(" Failure while starting server!");

            throw new Error(' Port is unavalible!');

        }

        this.app.use(this.getHeaders);

        this.app.use(ServerA.express.static(\_\_dirname + '/static'));

        this.app.post('/insert/record', this.insertRecord);

        this.app.post('/select/record', this.selectRecord);

        console.log(" Server started succesfully!");

    }

    getHeaders(request, response, next) {

        response.header("Cache-Control", "no-cache, no-store, must-revalidate");

        response.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

        response.header("Access-Control-Allow-Origin", "\*");

        next();

    }

    insertRecord(request, response) {

        function loadBody(request, callback) {

            let body = [];

            request.on('data', (chunk) => {

                body.push(chunk);

            }).on('end', () => {

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

                callback(body);

            });

        }

        loadBody(request, function(body) {

            const obj = JSON.parse(body);

            const name = obj.name;

            const price = obj.price;

            const storage\_path = "data/cars.json";

            const fd = ServerA.fs.readFileSync(storage\_path, "utf8")

            let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();

            const name\_exists = storage.has(name);

            let added = false;

            if (!name\_exists) {

                added = true;

                storage.set(name, price);

                ServerA.fs.writeFileSync(storage\_path, JSON.stringify([...storage]));

            }

            response.end(JSON.stringify({answer: added}));

        });

    }

    selectRecord(request, response) {

        function loadBody(request, callback) {

            let body = [];

            request.on('data', (chunk) => {

                body.push(chunk);

            }).on('end', () => {

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

                callback(body);

            });

        }

        loadBody(request, function(body) {

            const obj = JSON.parse(body);

            const name = obj.name;

            const storage\_path = "data/cars.json";

            const fd = ServerA.fs.readFileSync(storage\_path, "utf8")

            let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();

            let found = false;

            let price;

            if (storage.has(name)) {

                found = true;

                price = storage.get(name);

            }

            response.end(JSON.stringify({answer: found,

                                         price: price}));

        });

    }

}

class ServerB {

    static fs = require("fs");

    static express = require("express");

    constructor(port) {

        this.app = ServerB.express();

        this.port = port;

        try {

            this.app.listen(this);

            console.log(` Starting server on port ${this.port}... `);

        } catch (error) {

            console.log(" Failure while starting server!");

            throw new Error(' Port is unavalible!');

        }

        this.app.use(this.getHeaders);

        this.app.use(ServerB.express.static(\_\_dirname + '/static'));

        this.app.post('/insert/record' , this.insertRecord);

        this.app.post('/select/record', this.selectRecord);

        console.log(" Server started succesfully!");

    }

    getHeaders(request, response, next) {

        response.header("Cache-Control", "no-cache, no-store, must-revalidate");

        response.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

        response.header("Access-Control-Allow-Origin", "\*");

        next();

    }

    loadBody(request, callback) {

        let body = [];

        request.on('data', (chunk) => {

            body.push(chunk);

        }).on('end', () => {

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

            callback(body);

        });

    }

    insertRecord(request, response) {

        function loadBody(request, callback) {

            let body = [];

            request.on('data', (chunk) => {

                body.push(chunk);

            }).on('end', () => {

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

                callback(body);

            });

        }

        console.log(1);

        loadBody(request, function(body) {

            const obj = JSON.parse(body);

            const name = obj.name;

            const cars = obj.cars;

            const storage\_path = "data/storage.json";

            const fd = ServerB.fs.readFileSync(storage\_path, "utf8")

            let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();

            console.log(name, cars);

            console.log(storage);

            const name\_exists = storage.has(name);

            let added = false;

            if (!name\_exists) {

                added = true;

                storage.set(name, cars);

                ServerB.fs.writeFileSync(storage\_path, JSON.stringify([...storage]));

            }

            response.end(JSON.stringify({answer: added}));

        });

    }

    selectRecord(request, response) {

        function loadBody(request, callback) {

            let body = [];

            request.on('data', (chunk) => {

                body.push(chunk);

            }).on('end', () => {

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

                callback(body);

            });

        }

        loadBody(request, function(body) {

            const obj = JSON.parse(body);

            const name = obj.name;

            const storage\_path = "data/storage.json";

            const fd = ServerB.fs.readFileSync(storage\_path, "utf8")

            let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();

            let found = false;

            let cars;

            if (storage.has(name)) {

                found = true;

                cars = storage.get(name);

            }

            response.end(JSON.stringify({answer: found,

                                         cars: cars}));

        });

    }

}

class ServerC {

    static fs = require("fs");

    static express = require("express");

    constructor(port) {

        this.app = ServerC.express();

        this.port = port;

        try {

            this.app.listen(this.port);

            console.log(` Starting server on port ${this.port}... `);

        } catch (error) {

            console.log(" Failure while starting server!");

            throw new Error(' Port is unavalible!');

        }

        this.app.use(this.getHeaders);

        this.app.use(ServerC.express.static(\_\_dirname + '/static'));

        this.app.post('/add\_car', this.addCar);

        this.app.get('/get\_car', this.getCar);

        this.app.post('/add\_storage', this.addStorage);

        this.app.get('/get\_storage', this.getStorage);

        console.log(" Server started succesfully!");

    }

    getHeaders(request, response, next) {

        response.header("Cache-Control", "no-cache, no-store, must-revalidate");

        response.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

        response.header("Access-Control-Allow-Origin", "\*");

        next();

    }

    addCar(request, response) {

        const name = request.query.name;

        const price = request.query.price;

        function sendPost(url, body, callback) {

            const headers = {};

            const requests = require("request");

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

            headers["Connection"] = "close";

            requests.post({

                url: url,

                body: body,

                headers: headers

            }, function(error, response, body) {

                if (error) {

                    callback(null);

                } else {

                    callback(body);

                }

            });

        }

        sendPost("http://localhost:5001/insert/record",

                  JSON.stringify({name: name,

                                  price: price

    }), function(answerString) {

            response.end(answerString);

        });

    }

    getCar(request, response) {

        const name = request.query.name;

        function sendPost(url, body, callback) {

            const headers = {};

            const requests = require("request");

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

            headers["Connection"] = "close";

            requests.post({

                url: url,

                body: body,

                headers: headers

            }, function(error, response, body) {

                if (error) {

                    callback(null);

                } else {

                    callback(body);

                }

            });

        }

        sendPost("http://localhost:5001/select/record",

                  JSON.stringify({name: name}),

        function(answerString) {

            response.end(answerString);

        });

    }

    addStorage(request, response) {

        const name = request.query.name;

        const cars = request.query.cars;

        function sendPost(url, body, callback) {

            const headers = {};

            const requests = require("request");

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

            headers["Connection"] = "close";

            requests.post({

                url: url,

                body: body,

                headers: headers

            }, function(error, response, body) {

                if (error) {

                    callback(null);

                } else {

                    callback(body);

                }

            });

        }

        sendPost("http://localhost:5002/insert/record",

                  JSON.stringify({name: name,

                                  cars: cars}),

        function(answerString) {

            response.end(answerString);

        });

    }

    getStorage(request, response) {

        const name = request.query.name;

        const requests = require("request");

        function sendPost(url, body, callback) {

            const headers = {};

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

            headers["Connection"] = "close";

            requests.post({

                url: url,

                body: body,

                headers: headers

            }, function(error, response, body) {

                if (error) {

                    callback(null);

                } else {

                    callback(body);

                }

            });

        }

        sendPost("http://localhost:5002/select/record",

                  JSON.stringify({name: name}),

        function(answerString) {

            response.end(answerString);

        });

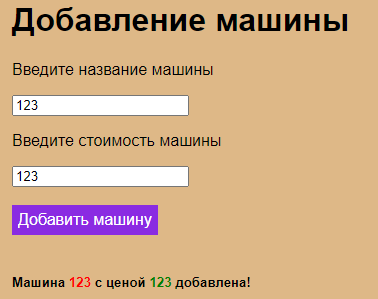
    }

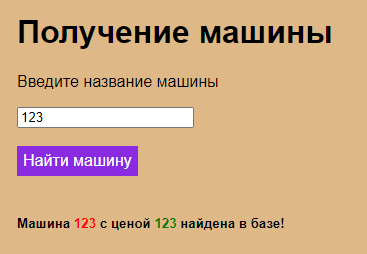
}

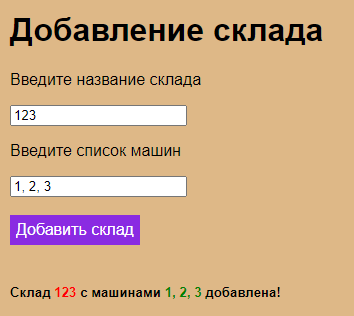
let serverA = new ServerA(5001);

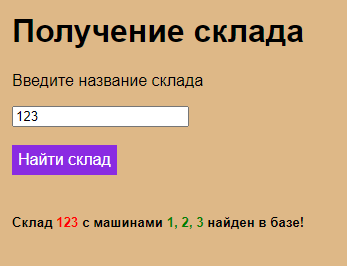
let serverB = new ServerB(5002);

let serverC = new ServerC(5003);











### 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**.

**Листинг:**

В index.js добавим строчки кода

"use strict";

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

// функция для вызова программы и получения результата её работы

function useCmd(s) {

    const options = {encoding: 'utf8'};

    const cmd = s.toString();

    const answer = execSync(cmd, options);

    return answer.toString();

}

// получаем параметры скрипта

const type = "" + process.argv[2];

if (type === "number") {

    const number = "" + process.argv[3];

    // получаем факториал числа

    const factorialCommand = `node number.js ${number}`;

    console.log(factorialCommand);

    let factorial = useCmd(factorialCommand);

    factorial = parseInt(factorial);

    console.log(factorial);

} else if (type === "array") {

    let count = "" + process.argv[3];

    count = parseInt(count);

    let element;

    let array = [];

    for (let i = 4; i < count + 4 ; i++) {

        element = "" + process.argv[i];

        array.push(element);

    }

    // получаем факториал числа

    const factorialCommand = `node array.js ${array}`;

    console.log(factorialCommand);

    let factorial = useCmd(factorialCommand);

    console.log(factorial);

} else {

    console.log("incorrect key");

}

Array.js

"use strict";

function factorial(n) {

    return n ? n \* factorial(n - 1) : 1;

}

let number = process.argv[2];

number = number.split(",");

let result = "";

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

    number[i] = factorial(parseInt(number[i]));

    result += number[i] + " ";

}

console.log(result);

Number.js

"use strict";

function factorial(n) {

    return n ? n \* factorial(n - 1) : 1;

}

const number = "" + process.argv[2];

const result = factorial(parseInt(number));

console.log("" + result);

****

****

### Задание 1

С клавиатуры считываются числа **A** и **B**. Необходимо вывести на экран все **числа Фибоначчи**, которые принадлежат отрезку от **A** до **B**.

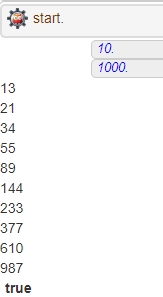
ok.

input(A, B) :- read(A), read(B); ok.

printA(A, S) :- A >= S, write(A), nl; ok.

cicle(A, B, S, F) :- C is (A + B), printA(A, S), B =< F, cicle(B, C, S, F); ok.

start :- input(A, B), cicle(1, 1, A, B); ok.

****

**Вывод:**

Все поставленные задачи были выполнены. Цель лабораторной работы достигнута.